AN ADJECTIVAL INTERFACE FOR PROCEDURAL CONTENT GENERATION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE,
FACULTY OF SCIENCE
AT THE UNIVERSITY OF CAPE TOWN
IN FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Carl Hultquist
December 2008

Supervised by
James Gain and David Cairns

Abstract

In this thesis, a new interface for the generation of procedural content is proposed, in which the user describes the content that they wish to create by using *adjectives*. Procedural models are typically controlled by complex parameters and often require expert technical knowledge. Since people communicate with each other using language, an adjectival interface to the creation of procedural content is a natural step towards addressing the needs of non-technical and non-expert users.

The key problem addressed is that of establishing a mapping between adjectival descriptors, and the parameters employed by procedural models. We show how this can be represented as a mapping between two multi-dimensional spaces, *adjective space* and *parameter space*, and approximate the mapping by applying novel function approximation techniques to points of correspondence between the two spaces. These corresponding point pairs are established through a training phase, in which random procedural content is generated and then described, allowing one to map from parameter space to adjective space. Since we ultimately seek a means of mapping from adjective space to parameter space, particle swarm optimisation is employed to select a point in parameter space that best matches any given point in adjective space.

The overall result, is a system in which the user can specify adjectives that are then used to create appropriate procedural content, by mapping the adjectives to a suitable set of procedural parameters and employing the standard procedural technique using those parameters as inputs. In this way, none of the control offered by procedural modelling is sacrificed — although the adjectival interface is simpler, it can at any point be stripped away to reveal the standard procedural model and give users access to the full set of procedural parameters. As such, the adjectival interface can be used for rapid prototyping to create an approximation of the content desired, after which the procedural parameters can be used to fine-tune the result. The adjectival interface also serves as a means of intermediate bridging, affording users a more comfortable interface until they are fully conversant with the technicalities of the underlying procedural parameters.

Finally, the adjectival interface is compared and contrasted to an interface that allows for direct specification of the procedural parameters. Through user experiments, it is found that the adjectival interface presented in this thesis is not only easier to use and understand, but also that it produces content which more accurately reflects users' intentions.

Acknowledgments

Many, many people need to be acknowledged for the successful completion of this thesis — too many to possibly remember or list in full! If I ever discussed my thesis with you, then you almost certainly had some impact — no matter how small — and I thank you for exercising patience, and listening to my ramblings.

First and foremost, I would like to acknowledge those who have sponsored this thesis financially. I was very fortunate to be sponsored in my first two years by the Harry Crossley Foundation in the form of a fellowship, and for this I am eternally grateful — without their assistance, those formative years would have been infinitely more stressful. The UCT Postgraduate Office, who administer the funding for the Harry Crossley Foundation, have also been extremely generous in their additional support through the Myer Levinson and Manuel & Luby Washkansky scholarships, and for this I am extremely thankful. In 2007, the National Research Foundation (NRF) provided assistance in the form of a scarce skills scholarship, and this is hereby acknowledged. The opinions expressed and conclusions arrived at, are mine and not necessarily to be attributed to the NRF.

The staff at both the University of Cape Town and the University of Stirling have been instrumental in both challenging and encouraging me along the way, and for their enthusiasm I am very thankful. Related to this, I owe Donald Cook, Peter Waker and the many other people involved with the South African Computer Olympiad a huge thank you — without the influence of the Olympiad I might never have studied (and kept studying) at UCT, and might not be where I am today.

My supervisors, James Gain and David Cairns, have been the cornerstone for the successful completion of this thesis. From being the source of great ideas, to keeping my mind diverted by involving me in their own research, to keeping my life balanced by showing me how professional academics play squash and windsurf, these two gentleman have been exceptional in their level of commitment to my progress as a student, and my development as a human being. I owe them both a great deal, and hope that over the years we will keep in touch and that I will have the opportunity to learn more from them.

Special mention needs to be made of Dave Nunez, Ilda Ladeira and Cara Winterbottom, three of my postgraduate peers. They provided me with some wonderful insight into experimental research, and were instrumental in the design and overall success of my user experiments — without which this

thesis would not have had such positive results! A huge thanks to you all — I hope that someday I can intellectually enlighten you in the same way as you have me.

Keeping with the university theme, my acknowledgements would not be complete without a special mention of my other friends and peers in the Computer Science Department — thank you all for being such wonderful friends and influences. In particular, a big thanks to Christopher de Kadt and Steve Asherson — without your support in the form of a daily game of DotA [IceFrog, 2000][1], I would have finished this thesis two years earlier. In a similar vein, a big thanks to Bruce Merry, for steering me off the path of gaming, and into the land of hacking on cool technology for the South Africa Computer Olympiad. Others outside the Computer Science Department also made my postgraduate experience extremely memorable — thanks to Arthur Rose and Grace Kendall for your daily distractions in the form of coffee and muffins in my office, or out and about around the campus. Sam Morar, thanks for the *many* late night playing backgammon, drinking wine, and musing over life, the universe and everything. It's been wonderful spending time with all of you in Cape Town, and I miss our times together.

A big thank you to everyone who took part in the experiments conducted as part of this research — without your assistance, this project would never have succeeded. Many thanks also to my housemates over the years — both in Cape Town and more recently in London — who have done well to put up with what may have seemed like the never-ending thesis. It's finally done, due in part to those of you who had faith in my abilities, and spurred me on to finish it. For this faith and encouragement, I am forever grateful. Special mention must be made of my London housemates — Laura Johnston, Kate Richardson and Mark Platt — who have put in a valiant effort to distract me and ply me with wine in the evenings! Without your kindly interventions I might have been finished a good 6 months earlier. It's been a wonderful year, and I look forward to more friendly banter in our garden without having to feel guilty about not tapping away at my keyboard. Huge thanks, too, to Hayley Podmore, for your encouragement and support through the final months of completing my corrections, and seeing me to the finish line!

By no means least, an extra-special thanks is owed to my family. To my good friends and extended family — the Hultquists, Voss's, Düsterwalds, Nelsons and Charlestons — thanks for the amazing times we've spent together. I'm very happy and extremely privileged to have you all in my life. To my grandmother, Gerda Voss (my Oma), thank you for always believing in me and seeing the best in me. To my brother, Marc, thank you for being my biggest supporter, and always sticking up for me. Lastly, to my parents, Adrian and Christl: you could not have done a better job as parents, and I'm really fortunate to have had your influence on, and guidance in, my life.

Finally, I dedicate this thesis to my good friend, Roland Craig. I wish you were still around to have seen me through the tough times, and laughed with me through the fun times. You'll always be missed.

- Carl Hultquist, 2008

[1] No, this isn't a real entry in my bibliography. But for those interested, see `http://www.dota-allstars.com`

Contents

Abstract **iii**

Acknowledgments **v**

1 Introduction **1**

1.1 Contributions 6

1.2 Thesis structure 7

2 Background to procedural modelling **9**

2.1 Basic methods 9

2.2 Application areas 14

2.2.1 Digital building (re)construction 15

2.2.2 Trees and plants 18

2.2.3 Terrain 20

2.2.4 Clouds, sea and skylight 23

2.2.5 Music and speech 25

2.3 Interfaces 26

2.3.1 Freehand sketching 26

2.3.2 Interfaces for parametrised procedural models 28

2.4 Summary 31

3 Overview of adjectival technique **33**

3.1 Motivation for adjectives, and related work 33

3.1.1 Related work in the use of adjectives 34

3.1.2 Related work on more general natural language interfaces 36

3.2 Principle of the technique 40

3.2.1 Modifications and extensions 42

3.2.2 Comparison and contrast to previous work 43

3.3 Summary 44

4 Background to function approximation **45**

4.1 Scattered data interpolation 46

4.1.1 Triangulation based methods 46

4.1.2 Natural neighbour interpolation 49

4.1.3 Inverse-distance based methods 52

4.2 Scattered data approximation 54

4.2.1 Least squares, weighted least squares, and moving least squares 54

4.2.2 Radial basis function networks (RBFNs) 58

4.3 Online learning 65

4.3.1 Artificial neural networks 66

4.3.2 Locally weighted projection regression 72

4.4 Summary 73

5 System architecture **75**

5.1 Direction of mapping 76

5.1.1 Genetic algorithms 77

5.1.2 Particle swarm optimisation 78

5.1.3 General comments on genetic algorithms and particle swarm optimisation . . 80
5.2 Properties of g . . . 81
5.2.1 Continuity . . . 81
5.2.2 Generalisation . . . 82
5.2.3 Locality . . . 82
5.2.4 Rapid evaluation . . . 83
5.2.5 Choice of function approximation technique . . . 83
5.3 Representation of adjectives . . . 83
5.4 Choosing subsets of adjectives . . . 86
5.5 Dynamic use of additional adjectives . . . 86
5.6 Augmenting training data with certainty values . . . 87
5.6.1 Applying certainty values to differing perceptions . . . 88
5.6.2 Applying certainty values to WordNet . . . 88
5.6.3 Incorporating certainty values into radial basis function networks . . . 89
5.7 Complete overview of technique . . . 93
5.8 Summary . . . 95

6 Experimental testing **97**

6.1 Experimental design . . . 98
6.1.1 First stage . . . 99
6.1.2 Second stage . . . 105
6.2 Objectives . . . 105
6.3 Hypotheses . . . 105
6.4 Results and discussion . . . 106
6.4.1 First stage . . . 106
6.4.2 Second stage . . . 113

6.5 Interpretation of results 114
6.6 Further evaluation 116
6.6.1 Tree generation 116
6.6.2 Emotive speech synthesis 118
6.7 Summary 119

7 Conclusions and future work **121**

7.1 Motivational summary 122
7.2 Future work 123
7.2.1 Better handling of non-continuous procedural methods 123
7.2.2 Conduct experiments with a more focused group of users 123
7.2.3 Hybrid personalised function training 124
7.2.4 Direct comparison to other state of the art interfaces 124
7.2.5 Online training 124
7.2.6 Local control 125
7.2.7 Natural language parsing 125
7.2.8 Use of non-parametric procedural inputs 125
7.2.9 Incorporating pre-defined content 126
7.3 Final thoughts 126

A First investigation: adjective representation and consistency of thought **127**

A.1 Outline of investigation 127
A.2 Objectives 132
A.3 Results and discussion 132
A.4 Summary 137

B Second investigation: large-scale data collection **141**

B.1 Outline of investigation 141
B.2 Objectives 142
B.3 Results and discussion 142
B.4 Summary 145

C Houdini procedural landscape system **149**

C.1 Terrain 153
C.1.1 Modelling the terrain 153
C.1.2 Shading the terrain 156
C.2 Rivers and lakes 158
C.3 Trees and scrubs 158
C.4 Putting everything together 160

Bibliography **161**

List of Tables

1 The number of simplices produced by Delaunay triangulation. 49

2 The number of coefficients required to specify a moving least squares system. 58

3 Inputs, outputs and constraints for the exclusive-or function. 67

4 Comparison of function approximation techniques. 85

5 The list of questions asked of users in the first stage of the experiment. 100

6 Responses from users of the direct specification interface in the first stage. 107

7 Responses from users of the adjectival interface in the first stage. 108

8 Responses from users of the direct specification interface in the first stage. 109

9 Responses from users of the adjectival interface in the first stage. 110

10 T-test results for scalar data. 111

11 T-test results comparing the data for the two tasks given to users. 112

12 Summary of combined accuracy data. 113

13 T-test result comparing combined accuracy data of users. 113

14 T-test results comparing time data. 113

15 Results of second stage experiment grouped by photograph. 115

16 Procedural parameters for our first investigation. 129

17 The parameter values used to generate environments for our first investigation. . . . 130

18 Results of various clustering metrics on experimental and random data. 134

19 Clustering ratios for experimental and random data. 135

20 Task settings for second investigation. 142

21 Iterations perform for each task in second investigation. 143

22 Values of the LOO estimator for data in the second investigation. 144

23 List of procedural parameters for final experiment. 151

24 List of procedural parameters for final experiment (continued). 152

List of Figures

1 The procedural construction of the Koch curve 3

2 Some examples of Julia sets 4

3 An example of three-dimensional Perlin and simplex noise [Gustavson, 2005]. 10

4 An example of Worley noise and its application. 11

5 An example of Perlin noise and fractional Brownian motion. 11

6 An example of applying fractional Brownian motion to Worley noise. 12

7 Some initial steps of the diamond-square algorithm. 12

8 An example of a plasma fractal generated by the diamond-square algorithm. 13

9 An illustration of the building reconstruction technique of Debevec et al. 16

10 An example of the automatic façade synthesis technique of Müller et al. [2007]. . . . 17

11 An example of the style grammar technique of Aliaga et al. [2007]. 17

12 An example of procedural modelling of buildings by Müller et al.. 17

13 An example of a city generated by the technique of Parish and Müller. 18

14 Examples of some extensions to basic L-systems. 19

15 An example reconstruction from the technique of Quan et al.. 20

16 Images from the work of Weber and Penn. 21

17 An example of a heightmap and its corresponding 3D terrain. 22

18 Blending together different types of noise. 22

19 An example of the terrain synthesis technique of Zhou et al. [2007]. 23

20 An example of how noise functions can be used to capture clouds and sea in nature. 24
21 Examples of cloud generation techniques. 25
22 An example of daylight as presented in Preetham et al. [1999]. 25
23 A tree created using the technique of Okabe and Igarashi. 27
24 An example of a design gallery 29
25 The interface of Merry et al. for prcoedural environment generation. 30
26 A graphical summary of the technique of Polichroniadis [2001]. 34
27 An example of a scene generated by WordsEye. 38
28 An example of the technique of Yanai and Okada [2006]. 38
29 Retrieving images from a database [Barnard and Forsyth, 2001]. 39
30 Automatically annotating images [Barnard and Forsyth, 2001]. 39
31 A point-set and its Delaunay triangulation 47
32 Computation of barycentric co-ordinates 47
33 The effect of adding a new point to a Voronoi diagram. 50
34 An illustration of the quantities used in the computation of the Laplace interpolant. 51
35 Varying the exponent of a Shepard interpolant. 53
36 The effects of gradient estimation in function approximation. 54
37 A diagram of a radial basis function network. 59
38 An approximation using a regression tree 64
39 Examples of activation functions used in neural networks. 67
40 The exclusive-or function of two variables, modelled using two threshold logic units. 68
41 A schematic for a multilayer perceptron. 68
42 An example of how overfitting can occur. 69
43 Illustration of the technique of Vijayakumar et al. [2005]. 73
44 An illustration of the overall system proposed by this thesis. 76

45 Methods for crossover in genetic algorithms. 78

46 Examples of procedural landscapes generated for final experiment. 98

47 Photographs of real-life landscapes presented to users for the experiment. 101

48 Interface allowing for the direct specification of procedural parameters. 102

49 Adjectival interface for choosing an adjective. 103

50 Adjectival interface for adjusting the quantification and negation of adjectives. . . . 104

51 Examples of trees generated using an adjectival interface. 117

52 Some procedural environments generated for our first investigation. 128

53 The user interface for capturing descriptions in our first investigation. 131

54 Visualisations of the data collected for the first investigation. 133

55 Clustering ratio histograms for our first investigation. 136

56 2D visualisations of the data collected in the first investigation, environments 1 and 2. 137

57 2D visualisations of the data collected in the first investigation, environments 3-7. . . 138

58 2D visualisations of the data collected in the first investigation, environments 8-12. . 139

59 2D visualisations of the data collected in the first investigation, environments 13-15. 140

60 Distribution of iterations completed in second investigation. 143

61 Box and whisker plots for second investigation. 144

62 Distribution of response for the *wet/dry* descriptor in the second investigation. . . . 145

63 Distribution of response for the *sparse/lush* descriptor in the second investigation. . 146

64 Distribution of response for the *tropical* descriptor in the second investigation. 146

65 Distribution of response for the *cloudy* descriptor in the second investigation. 147

66 Distribution of response for the *light/dark* descriptor in the second investigation. . . 147

67 Distribution of response for the *mountainous* descriptor in the second investigation. . 148

68 Distribution of response for the *undulating* descriptor in the second investigation. . . 148

69 An example of a procedural network in Houdini. 150

70 Compositing networks used to create an escarpment. 154

71 Compositing networks used to create a central highlands area. 155

72 An example of the final heightmap combining large- and small- scale features. 155

73 An example of assigning zones to the terrain. 156

74 An example of shading of the terrain. 157

75 The instances of vegetation used for desert regions. 158

76 The instances of vegetation used for green zone regions. 159

77 The instance of vegetation used for beach regions 159

78 Examples of a fully rendered landscape. 160

Chapter 1

Introduction

In today's world of fast-paced technology, the performance of central processing units and graphics processing units is increasing at an incredibly rapid rate. The most well known measure of this is *Moore's Law* [Moore, 1965], which posits an exponential increase in the number of transistors that can be placed on an integrated circuit. Since Moore's initial observations, this exponential trend has also been observed in other aspects of technology such as computing power per unit cost, and the rate of increase in hard drive and RAM storage capacities. With all this rapid technological innovation, the modern home computer is becoming much more capable of presenting ever larger and more complex digital content — for example, virtual environments, which are used extensively in computer games and simulations. There is a corresponding demand to leverage this by creating larger and more complex digital content that pushes the limits of the hardware available — in the case of virtual environments, more complex environments can achieve greater realism and allow the user to feel more "present" in the environment.

To complicate the situation, technological advances have driven a general speedup in service delivery, causing consumers to become more accustomed to quicker turnaround times and putting pressure on companies to at least maintain, or improve, their rate of delivery. To exacerbate the problem, increased competition resulting from a lowering of industry entry boundaries causes even more pressure in the form of tighter deadlines and more competitive service costs. Combining this with the need and the pressure to create more intricate content, causes a serious problem: humans undergo a much slower "hardware" evolution, and simply cannot keep up with the rate at which technology is advancing nor the demands that stem from that advancement.

Point 1: ***Humans cannot keep up with the rate of technological development.***

There are two fairly intuitive means of dealing with this problem: hire more employees, or leverage the new technology in a way that makes it easier for people to perform more complex tasks. Global trends in employment do indicate an increase in the number of people employed over time [ILO, 2007], although this cannot be solely attributed to the need for keeping up with technological development.

Ultimately, any organisation will reach a point of diminishing returns where it becomes infeasible to hire more employees. Taking advantage of the increased computational power must therefore also play a part in solving the problem.

This in itself is not a trivial step. A significant portion of software development over the past decade has been in the creation of increasingly higher level interfaces to the underlying hardware. This has been driven by two factors: in part by industry and the need there for employees to perform complex tasks more quickly; and in part by the explosion of technology in the home, and the need there for interfaces which are simple and easy to use by non-professional home users. The outcome of all this high-level interface simplification is what is known as *Wirth's Law* [Böszörményi et al., 2000], which states that "software gets slower faster than hardware gets faster". Although hardware efficiency has increased it has largely retained its simplicity, and so increasingly complex layers of software lead to greater quantities of interpretation that must be performed in order to communicate with the relatively simplistic language of hardware.

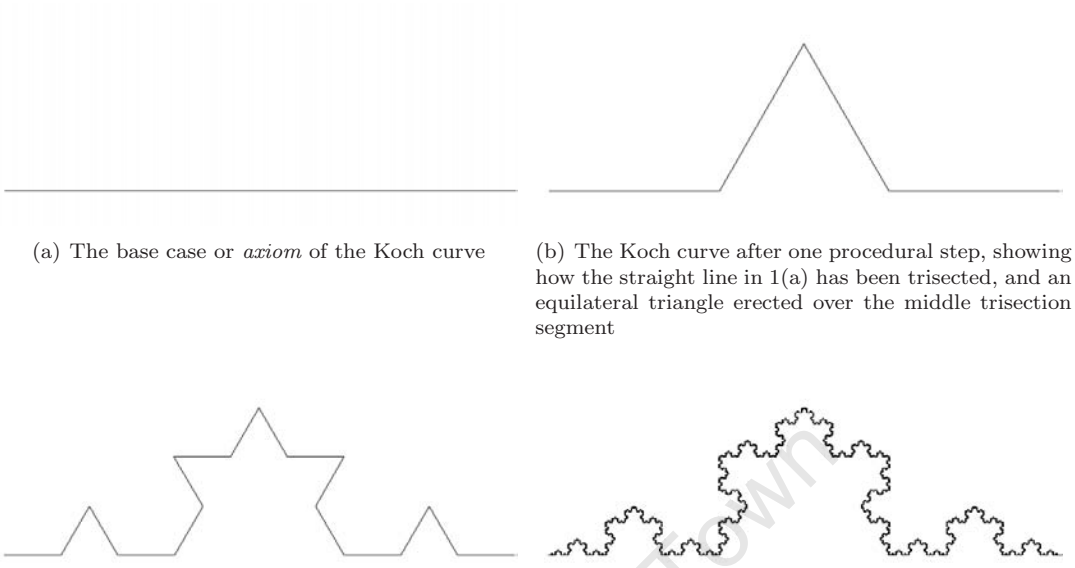
Instead of improving the usability of software, an orthogonal approach is to automate processes that might previously have been done by a human. In the realm of digital content generation, such techniques are referred to as *procedural methods*, and provide a means for machines to do what they do best — tediously processing a set of instructions — and freeing up human resources for more interesting tasks. As a simple example, consider the Koch curve [von Koch, 1906; Mandelbrot, 1983] shown in Figure 1. The curve is generated by repeatedly applying a pre-defined rule, which in this case is to trisect every line segment and erect an equilateral triangle over the middle trisection segment. Whilst this action could be performed by a human to some desired level of complexity, the task is easily expressed as a set of parameters and rules which can then be automated. In this case, the parameters of the model are the starting configuration (or base case) and the number of times which the rule should be applied. Furthermore, an automated approach will not only be able to perform the task more quickly than a human, but with more accuracy and complexity.

Another classical example of a procedural method is the *Julia set* [Falconer, 2003], and in particular Julia sets that arise from the *Mandelbrot set*. We present here a quick overview of these Julia sets; a more rigorous definition and other details can be found in Falconer's book. Suppose we have a quadratic polynomial $f_c : \mathbb{C} \to \mathbb{C}$ such that

$$f_c(z) = z^2 + c$$

where c is a fixed, complex parameter. The Julia set of the point c is then the set of points z which give rise to a chaotic sequence $f_c(z), f_c^2(z) = f_c(f_c(z)), f_c^3(z), \ldots, f_c^n(z), \ldots$ as $n \to \infty$; chaotic in the sense that the sequence neither diverges to infinity nor converges to some fixed value. By varying c and using colour to indicate the speed at which divergent z values escape to infinity, a vast set of beautiful and complex images can be created (see Figure 2).

Whilst the Koch curve and Julia sets are somewhat simplistic constructions, they illustrate well the



(a) The base case or *axiom* of the Koch curve

(b) The Koch curve after one procedural step, showing how the straight line in 1(a) has been trisected, and an equilateral triangle erected over the middle trisection segment

(c) The Koch curve after two procedural steps, showing how the same procedural rule has now been applied to all edges in 1(b)

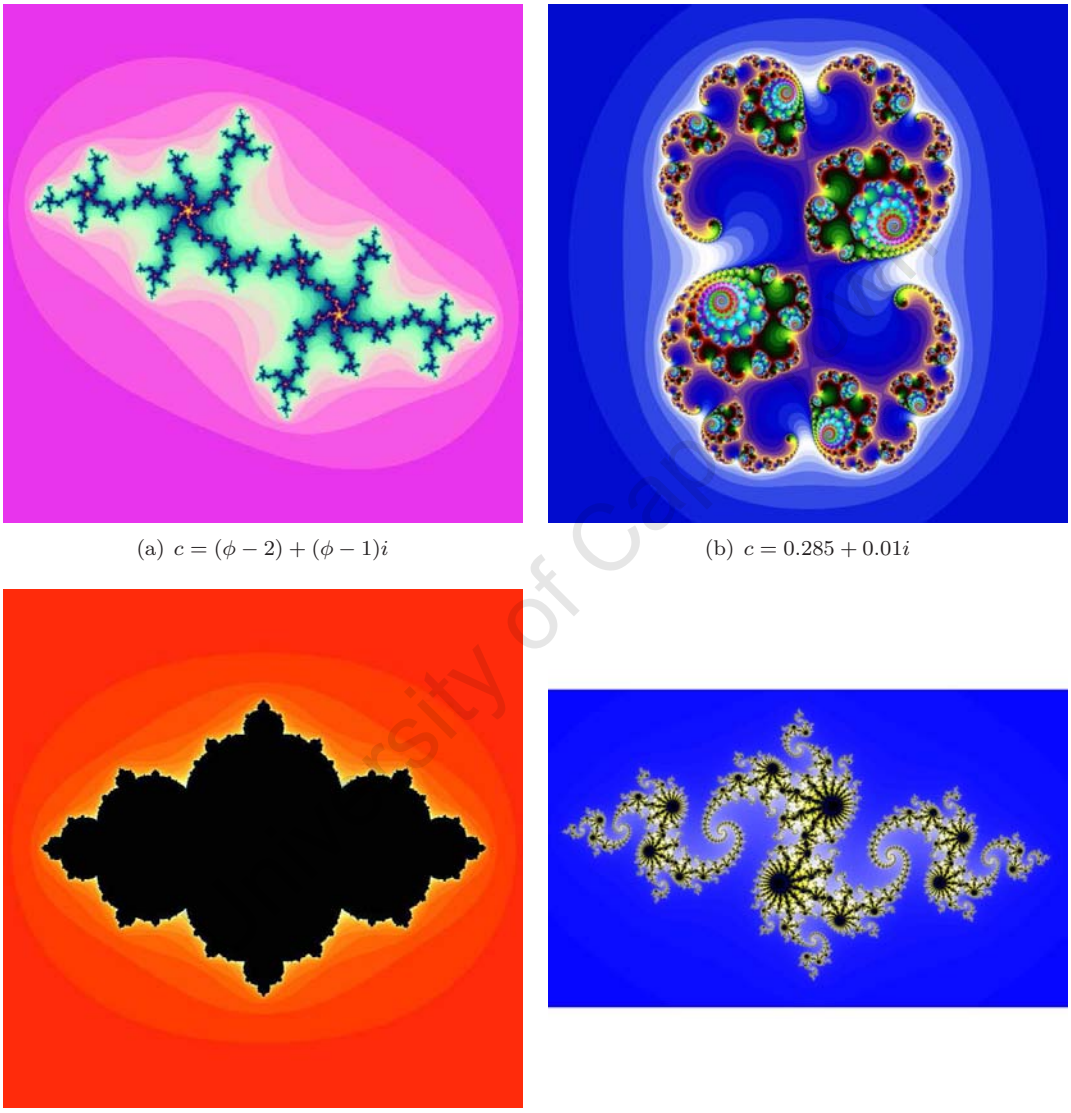
(d) The Koch curve after many procedural steps.

Figure 1: *The procedural construction of the Koch curve*

concepts behind procedural models — namely that by embodying tedious computation within an automated procedure, a variety of complex constructions can be created by merely modifying simple parameters.

There is, however, quite a large trade-off for this interface in the form of fine control over the output of a procedural model. Numeric parameters do not necessarily provide a useful interface to all users, and typically the nature of the parameters is such that a knowledge of the underlying procedure is required in order to fully understand how the parameters affect the resulting output. This is particularly evident when considering home users: with the advent of powerful home computing, tools that were previously reserved for usage in industry have become available to the average consumer. Whilst this is good in that it affords home users the opportunities to create their own content, in most cases the users lack the required technical knowledge or training to properly use these tools. Consider again the Julia sets: members of the mathematical community, who are intimately acquainted with the underlying mathematics of Julia sets and their relationship to the Mandelbrot set, would likely be able to predict with reasonable accuracy what the Julia set corresponding to a particular value of c would look like. Or given an image showing a Julia set, they could give educated guesses on the nature of the c value that generated that image. The average home user, however, would be at a complete loss — it is not at all obvious from the examples shown in Figure 2 how the c values relate to the images shown. Yet there *is* a scientific explanation, tied strongly to the formal definition of a Julia set and to the exact procedure with which the images were created.

Combining this unintuitive interface with our earlier discussion of the drive towards more complex



(a) $c = (\phi - 2) + (\phi - 1)i$

(b) $c = 0.285 + 0.01i$

(c) $c = 1 - \phi$

(d) $c = -0.8 + 0.156i$

Figure 2: *Some examples of Julia sets generated with various c parameters. In 2(a) and 2(c),* $\phi = \frac{1+\sqrt{5}}{2}$ *is the golden ratio.*

content as a result of greater processing power, means that home users are even less able to keep up in the creation of compelling content.

Point 2: ***Whilst procedural models provide a means for leveraging greater computational power without a corresponding need for greater man-power, their parametric interface inhibits their usage by non-technical users.***

The second point above is worth dwelling on, and provides the major motivation for our work. Since tools for creating digital content were first developed for use by trained professionals in industry, little work has been done to focus on the usability of these tools by non-technical or novice users. At the same time, there is a need to maintain the high technical capabilities available to advanced users, and simplifying the tools can in many instances limit their power and flexibility.

With these thoughts in mind, a technique that addresses the problems raised should provide the following features:

1. Allow large and complex procedural content to be created quickly.
2. Provide an interface that is usable by novice and non-technical users.
3. Maintain the flexibility afforded by parametrised procedural models.

In this thesis, a new technique is presented that allows the user to generate procedural content using *adjectival descriptors*. A brief overview of the technique is as follows:

1. **Modelling:** An existing procedural system is used to generate random samples of digital content, each of which is described using adjectives by an expert user or designer. The descriptions are used to train a set of extended radial basis function networks, which together provide a means for mapping a set of procedural parameters to an adjectival description of the content created by the procedural parameters.
2. **Usage:** A user specifies adjectives to describe the content that they wish to generate. A particle swarm optimisation algorithm is executed to search the space of all procedural parameters, finding the parameters that map as closely to the user's description as possible. The procedural system is then engaged to create content, and present this to the user.

The interface is thus presented as an additional layer of abstraction that establishes a mapping between adjectives and the underlying procedural parameters, and provides the features desired:

1. Existing procedural models are employed "under the hood", and these already provide for quick generation of complex output as will be explored in Chapter 2.
2. All people communicate with language, using adjectives to guide their descriptions of objects and occurrences. An adjectival interface should thus be readily usable by all users — be they technical or non-technical.

3. As the technique presented is implemented as an abstract layer on top of the procedural parameters of the model, it provides a form of *intrinsic scaffolding* [Jackson et al., 1998] in that the adjectival interface can be used for initial generation of content, and further minor modifications can be made afterwards by the user at the procedural parameter level. The adjectival interface can also be seen as an intermediate bridging method, providing an easy means for a user to generate content until they are fully conversant with the underlying procedural parameters.

Once the technique has been presented and its implementation discussed, it is compared and contrasted with the alternative of having direct control over procedural parameters. Whilst the focus is on procedural models that make use of a parametised interface, we also realise that alternative forms of input to procedural models are available and discuss these in Chapter 2.

1.1 Contributions

The contributions of this thesis are:

- A novel adjectival interface for the generation of procedural content is presented. The advantages and disadvantages of such an interface are discussed, and solutions presented for problematic aspects of the system. In particular, the representation of adjectives and the precise means of mapping from adjectives to procedural parameters are thoroughly analysed, and appropriately implemented.
  - Three problem domains are targeted to validate the adjectival interface — *virtual landscape generation*, *tree generation* and *emotional speech synthesis*.
  - The adjectival interface is shown to be more usable than one offering direct control over procedural parameters.
  - The adjectival interface is shown to produce content that more accurately reflects the intentions of the user, than one offering direct control over procedural parameters.
- An extension of radial basis function networks is introduced, allowing *certainty* values to be associated with the training data. These certainty values reflect how trusted each data point is, and allow certain data to have more influence on the final function approximation. It is shown that certainty values can be easily incorporated into a regular radial basis function network without affecting the underlying mathematical framework, to give a *certainty radial basis function network*.

1.2 Thesis structure

The structure of the thesis and a brief overview of the content of each chapter is as follows:

- **Chapter 2:** We begin by exploring the background of procedural modelling, with a specific focus on the modelling of computer graphics content and ranging from the modelling of individual objects to the synthesis of entire environments.
- **Chapter 3:** We motivate and introduce our technique, considering related work that has been done in other fields and how it compares or contrasts with our work.
- **Chapter 4:** Building on the overview of our technique given in Chapter 3, we posit the core problem as multidimensional scattered data approximation. We explore the technical challenges and details of this formulation with reference to related literature.
- **Chapter 5:** Taking into account specific considerations of our application, we present extensions to the function approximation techniques discussed in Chapter 4, and give a detailed description of our system as a whole.
- **Chapter 6:** We present and discuss an experimental design to test the effectiveness of our technique, and evaluate the results of our testing.
- **Chapter 7:** A final summary is presented, and conclusions drawn on the results of our experimentation. Ideas for future work and expansion on the technique are also presented and discussed.

Chapter 2

Background to procedural modelling

The key idea behind procedural modelling is that, given a small number of user inputs, a set of rules amplifies those inputs to produce output that is typically quite complex in nature. We already saw examples of this in Chapter 1, in the form of the Koch curve and Julia sets. Although a human could follow the same rules to create the final output, this is a mechanical process and so is better done by a computer than by a human designer.

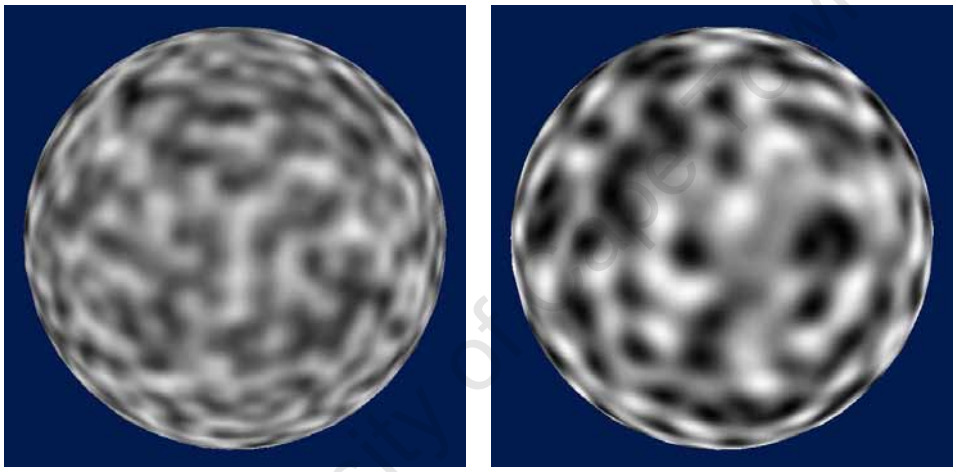
A significant quantity of work has been done in the field of procedural modelling, which we will cover in three sections that progressively build upon each other: *basic methods*, *application areas* and *interfaces*. For a general background to procedural modelling, its advantages and its issues, the reader should consult the work of Ebert et al. [1994].

2.1 Basic methods

Underpinning much of the modern development in procedural modelling are a number of core techniques, which have become such an intrinsic part of procedural modelling that they are often neglected as procedural models in their own right. One of the most commonly used groups of procedural methods are those that generate *noise*. Objects in the real world exhibit natural imperfections, and to mimic these in digital models, geometric and texture noise are often added. A simplistic way to generate noise is through random perturbations, but this suffers from the possibility of visually jarring discontinuities. *Perlin noise* [Perlin, 1985, 2002; Ebert et al., 1994] addresses this by smoothly interpolating across an integer lattice of random values, producing noise which is continuous and thus better models noise apparent in the real world. Procedural parameters controlling the frequency and amplitude allow for characteristics of the noise to be altered, and by offsetting all calculations,

different patches of noise with the same characteristics can be generated.

Since his initial publication, Perlin went on to improve his algorithm [Perlin, 2002] and as a need for noise in higher dimensions arose (for example, in the creation of animated volumetric textures) he proposed a new method for creating noise: *simplex noise* [Perlin, 2001; Gustavson, 2005]. The key difference in simplex noise is that it interpolates between the vertices of an n-dimensional *simplex*[1], in contrast to Perlin noise which interpolates over the vertices of an n-dimensional hypercube. This not only has computational benefits (an n-dimensional simplex has n vertices, in comparison to the 2^n vertices of an n-dimensional hypercube) but also gives rise to what is known as an *isotropic* texture — that is, a texture which has no discernable directional artifacts. Figure 3 shows a side-by-side comparison of three-dimensional Perlin and simplex noise.



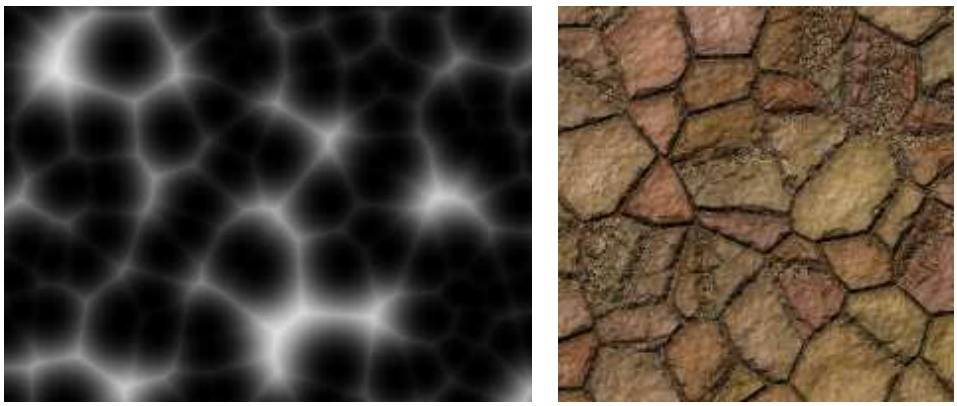
(a) Perlin noise (b) Simplex noise

Figure 3: *An example of three-dimensional Perlin and simplex noise [Gustavson, 2005].*

Although Perlin noise, and more recently simplex noise, have been the techniques most often used, these are not the only forms of noise generator that are useful. Another popular class of noise functions are those which generate *cellular* noise. In contrast to the somewhat cloudy appearance of Perlin and simplex noise, cellular noise functions exhibit a distinct cellular structure — see for example Figure 4(a) which shows a sample of *Worley noise* [Ebert et al., 1994]. Also referred to as *Voronoi noise*, Worley noise is generated by randomly scattering a number of seed points in space and computing the noise at any point as a function of the distance to the nearest seed point. Such noise functions are useful for creating natural or man-made elements that exhibit cellular structure, such as the cobblestones shown in Figure 4(b).

Typically, noise is used to generate 2D textures that are applied to 3D models. This can, however, cause the noise to appear stretched due to distortions in the parameterization of the surface. Recent developments such as *wavelet noise* [Cook and Derose, 2005] and *anisotropic noise* [Goldberg et al., 2008] provide a means for overcoming this, and give results that more closely resemble the uniform

[1] An n-dimensional simplex is the n-dimensional analogue of a 2-dimensional equilateral triangle

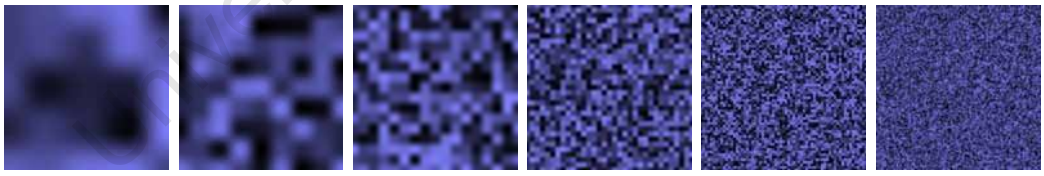


(a) (b)

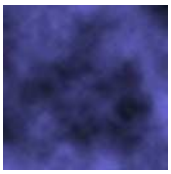
Figure 4: *An example of Worley noise (a), and its application in creating cobblestones (b).*

noise achieved through solid texturing.

Whilst useful for providing the random perturbations that make objects appear more realistic, noise functions are still lacking in one aspect of realism in that they can be too regular. One common application for noise is the modelling of a landscape heightfield. Consider how a real world landscape looks: there is noise on a very coarse level (giving rise to plateaus and mountain ranges), as well as noise on successively finer levels (giving rise to smaller mountains, hills, and eventually minor bumps and ditches). Multiple layers of noise, at varying frequencies and amplitudes, can be blended together in a technique known as *fractional Brownian motion* (fBm) to give a more realistic result (see Figures 5 and 6).



(a) Perlin noise at various frequencies



(b) Combination of the above layers of Perlin noise

Figure 5: *An example of Perlin noise and fractional Brownian motion.*

Another means for overcoming the regularity in Perlin and simplex noise is through the use of fractals, which we have already seen as good examples of procedural techniques. One such technique is the *diamond-square* algorithm [Fournier et al., 1982; Miller, 1986], which was first proposed for two dimensions but can also be extended to higher dimensions. Starting with a square domain

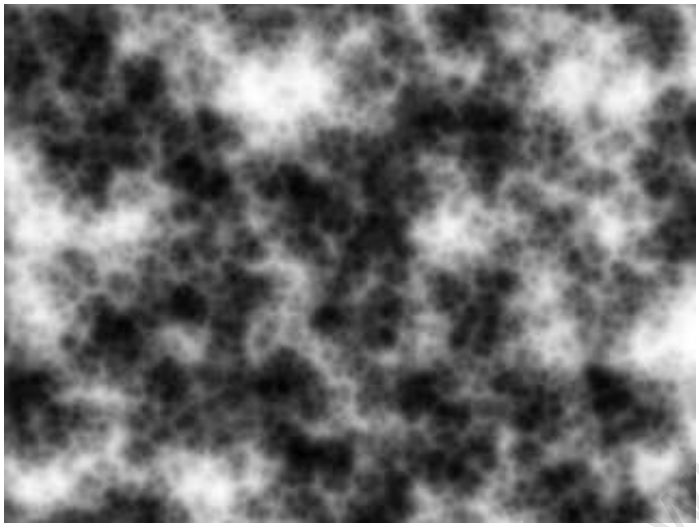


Figure 6: *An example of applying fractional Brownian motion to Worley noise.*

where the noise is defined at the corners (either by the user or randomly), the algorithm proceeds to subdivide the area repatedly in two successive steps:

1. **Square step:** For each square, calculate the noise at the square's midpoint by averaging the noise at the corners and adding a random perturbation (Figures 7(a) and 7(c)). This gives rise to a number of diamond shaped areas (Figures 7(b) and 7(d)), to which the diamond step is applied.
2. **Diamond step:** For each diamond, calculate the noise at the diamond's midpoint by averaging the noise at the corners and adding a random perturbation. This gives rise to a number of square shaped areas to which the square step is applied.

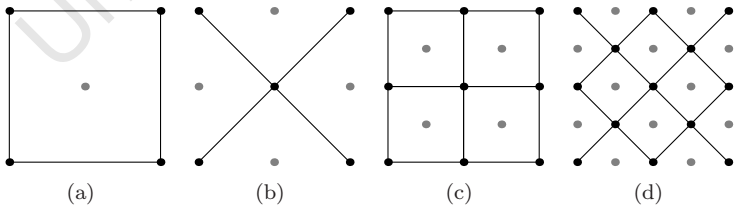


Figure 7: *Some initial steps of the diamond-square algorithm. At each stage, points in solid black reflect those for which noise values have already been calculated, and those in gray are points whose noise values are about to be calculated. The solid lines indicate the boundaries of the squares and diamonds referred to in the steps above.*

Repeating this process gives successively finer detailed noise, to any level of accuracy desired. The random perturbation applied at each step is typically scaled by the size of the diamond or square, and the magnitude of the unscaled perturbation affects how chaotic the resulting noise is. Rendered as a two-dimensional image, this algorithm generates a plasma effect and is often referred to as the *plasma fractal* or *random midpoint displacement fractal* (see Figure 8).

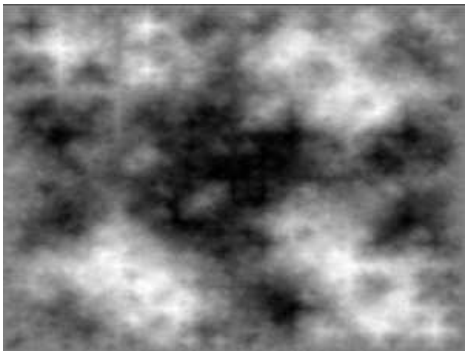


Figure 8: *An example of a plasma fractal generated by the diamond-square algorithm.*

Overcoming the regularity of other forms of noise comes at a price, though — noise generated using the diamond-square algorithm is nowhere continuous, which in many cases is undesirable. Without dismissing fractals completely, however, this exact property is in some cases useful as it allows for infinite detail — zooming in on any part of the noise will retain the inherent "noisyness" of the original. It also allows for a quick and coarse approximation to be quickly computed, which makes for a more interactive user interface to procedural modelling.

Closely related in the sense that they allow for progressive development of a model are *rule-based* systems, such as *Lindenmayer systems* [Lindenmayer, 1968; Prusinkiewicz and Lindenmayer, 1990; Měch and Prusinkiewicz, 1996] and *split grammars* [Wonka et al., 2003]. Rule-based systems are another well used core procedural element with the unique property that they allow for parallel research and novel development in two distinct areas. Extensions and alterations can be made to the underlying rule processing system, and then actual rule sets that achieve new results can be independently developed. The earliest significant contribution to rule sets can arguably be attributed to Lindenmayer [Lindenmayer, 1968], after whom *Lindenmayer-systems* or *L-systems* are named. Lindenmayer sought to replicate the beauty of nature on a computer, and his concept of L-systems achieves this by mimicking the processes of biological development. If one takes some string — analogous to the DNA or coding for an object — then an L-system determines how that string grows and changes over time, subject to its contents. This is closely linked to biology, where cells replicate and are modified subject to their state and the state of their surrounding cells. The initial string is referred to as the *axiom* string, and rules in an L-system have the following form:

$$\rho_n \quad : \quad module \rightarrow replacement$$

where *module* refers to a piece of genetic code to be replaced, and *replacement* refers to the replacing code. An L-system can have any number of rules, and if more than one rule matches a piece of genetic code then the first rule in the grammar is usually applied. It should be noted that L-systems are a *parallel string-rewriting system*, in that all the replacements happen simultaneously or in parallel.

Consider a system such as the one below

$$\begin{aligned} \omega &: \text{AB} \\ \rho_1 &: \text{A} \rightarrow \text{xA} \\ \rho_2 &: \text{B} \rightarrow \text{By} \end{aligned}$$

where ω indicates the axiom string. Rule ρ_1 matches the A in the axiom, and rule ρ_2 matches the B, so after one iteration the axiom is rewritten in parallel to give the string xABy. Once a prescribed number of iterations have been applied, the resulting string is interpreted in order to generate a model. One of the most common interpretations is as a set of *turtle* commands, which is covered in extensive detail by Prusinkiewicz and Lindenmayer [1990].

Many extensions to this basic rule system have been developed, the most prominent of which extends L-systems to be both stochastic and context-sensitive [Prusinkiewicz and Lindenmayer, 1990]. In such systems, rules are modified to the general form below:

$$\rho_n \quad : \quad \textit{left-context} <\textit{module}> \textit{right-context} \rightarrow \textit{replacement} : \textit{probability}$$

Such a specification gives much greater power to an L-system, by allowing rules to be applied only in particular circumstances and also to add a stochastic nature to the process — both of which are more in keeping with biological processes as we understand them. Building on this, further work was done by Měch and Prusinkiewicz [1996] to develop the concept of an *open L-system*. This allows for a form of two-way communication between the developing model and an outside environment, allowing the model to influence its environment and allowing the environment to affect changes and specific development in the model.

Furthermore, L-systems are not the only rule-based systems in use. Another example is the *split grammar*, introduced by Wonka et al. [2003]. Split grammars are a specialised form of set grammars, which are a mathematical means of describing shape primitives using sets, and defining rules as functions that transform one set into another. Split grammars extend set grammars by additionally allowing rules that split shape primitives.

2.2 Application areas

Having looked at some examples of basic procedural methods, it is not immediately clear how these might be used to create useful content. There are, however, many applications, which are now explored in more detail.

2.2.1 Digital building (re)construction

Creating models that capture a specific instance of a real life object can be a tricky process. One common example is in the digital reconstruction of environments, and in particular buildings. Such reconstructions allow people to "explore" a building without needing to be physically present, and a successful technique would have applications in a wide variety of fields such as engineering, architecture, geomatics and archaeology. The seminal work of Debevec et al. [1996] addresses this problem through a hybrid approach, making use of both photographs of the building being reconstructed, as well as a simple user-specified model. In particular, the user specifies the building model using a number of simple parametric primitives — such as cuboids and wedges — whose parameters (such as width and height) are free variables solved for by the system (Figure 9(a)). Additionally, constraints can be placed on the primitives: for example, forcing certain edges to have the same length, or forcing faces of different primitives to be joined together and thus cause the primitives to scale and move with each other (Figure 9(b)). The user is also required to draw lines on the photographs and indicate the corresponding edges in the model (Figures 9(c) and 9(d)). Using these correspondences their system is able to solve for the model parameters and calculate view-dependent texture information for the model from the photographs.

Although this reconstruction technique is extremely effective and allows models of buildings to be fairly easily reconstructed, the resulting model is still quite simplistic and does not capture geometric detail that is often present in real buildings, such as friezes or cornices. The authors go on to describe a means of *model-based stereopsis*, in which separate photographs that view some common part of the model can be used to infer how the actual scene deviates from the approximate model.

An alternative method for capturing the details of façades is presented by Müller et al. [2007]. From an input photograph of the façade of a building, their method automatically subdivides the image into floors and tiles containing basic primitives such as doors, windows and ledges, drawing on the splitting rules of shape grammars [Müller et al., 2006; Wonka et al., 2003]. Their algorithm procedes to summarise this tile data in what they dub the *irreducible façade*, which is a reduced image that encodes the core differences of the overall façade by removing symmetrical data. The tiles in the irreducible façade are then further subdivided to find the various structural components. This hierarchical subdivision is matched against a library of architectural elements, and used to select a high-detail geometric model that will replace the tile in the output geometry. The hierarchical subdivision of the original façade image is also used to automatically create a shape grammar from which the resulting building is synthesised. Finally, the original photograph is used to texture the resulting model. An overview of this process can be seen in Figure 10.

For the stylised synthesis of individual buildings, Aliaga et al. [2007] offer a solution that builds upon the techniques of Debevec et al. [1996] and Müller et al. [2006]. After applying the photogrammetric method of Debevec et al. to photographs of a building, their algorithm extracts a shape grammar that captures the building by finding repetitive patterns in the model. Given the primitive model for a new building, they match rules in the shape grammar to the model and in so doing transfer

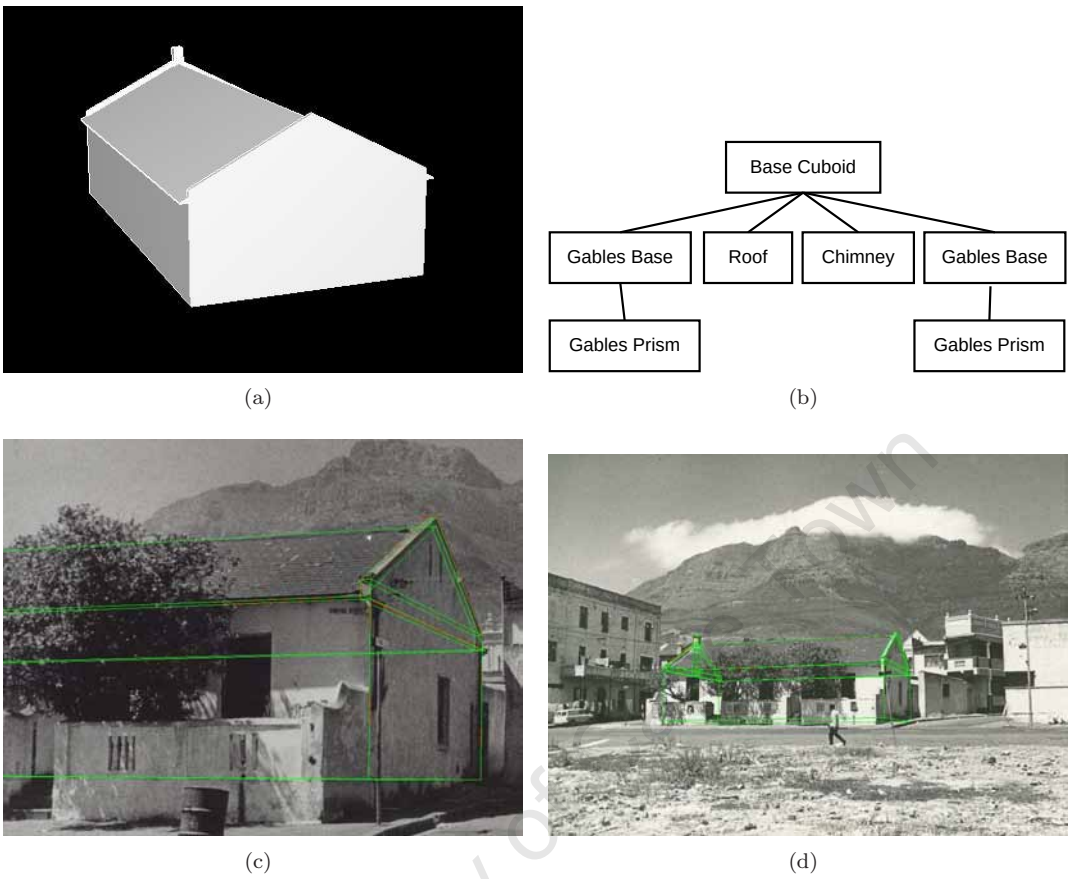
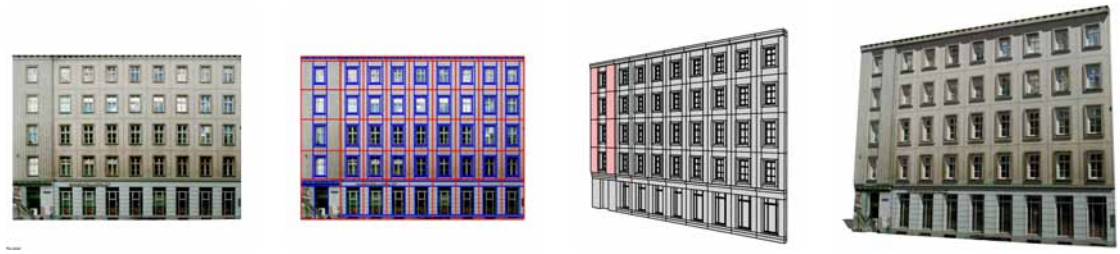


Figure 9: *An illustration of the building reconstruction technique of Debevec et al. [1996]. (a) shows the basic primitive model used, with the relationships between the individual primitives captured by the scene graph in (b). (c) and (d) are photographs of the building to be digitially reconstructed, with green lines overlayed in correspondence with edges of the primitive model. Images courtesy of de Kadt [2007].*

the style of the original building to the new one (see Figure 11).

Parish and Müller [2001] extend the modelling of individual buildings to an entire city, by making use of extended L-systems in order to grow a street map for a virtual city. Their extension allows for the parameters of modules in the system to be specified by external functions rather than rules in the system itself. This enables more complex calculations such as constraining the street network to avoid steep gradients and water areas. Thus, the L-system is used to generate what would be the ideal road network, and the external functions manipulate this according to the constraints inherent in the terrain. In this way, the technique is able to harness both the usefulness of L-systems in creating rule-based patterns, but also the power of other techniques to better analyse the validity of the output with regard to external constraints. In addition to this, the authors make use of self-sensitivity in their L-system, in that streets need to be aware of each-other in order to generate intersections. *Image maps* are extensively used as a form of alternative input to guide the L-systems, which the authors class into two categories: *geographical maps* and *sociostatistical maps*. Geographical maps



(a) (b) (c) (d)

Figure 10: *An example of the automatic façade synthesis technique of Müller et al. [2007], showing (a) the input photograph (b) the hierarchical split into tiles (c) the inferred three-dimensional model and (d) the final textured result.*

(a) (b) (c) (d)

Figure 11: *An example of the technique of Aliaga et al. [2007], in which the style of a building (a) is captured (b-c) and transferred onto the model of a new building (d).*

are used to specify elevation, land, water and vegetation data, whilst sociostatistical maps control aspects such as population density, zoning, street patterns and maximal building height.

Müller et al. [2006] approach digital building and city construction from a different angle, by employing rule systems to synthesise new architecture. By combining aspects from the works of Parish and Müller [2001] and Wonka et al. [2003], they introduce a shape grammar for CG architecture that is extremely similar to an L-system in its parallel rewriting structure, but which also incorporates aspects of split grammars to allow for primitives to be divided and analysed in smaller components. One of the core realisations of the authors is that L-systems, as used in botanical modelling, very effectively capture growth of objects over time — a concept that does not suit buildings in their entirety particularly well. However, it is suited to buildings that are constructed in various stages, such as the erecting of the façade or the addition of a roof. Thought of differently, the construction of a building can be divided into various stages, each of which adds successive detail

Figure 12: *An example of a wealthy suburbia environment, generated by the technique of Müller et al. [2006].*

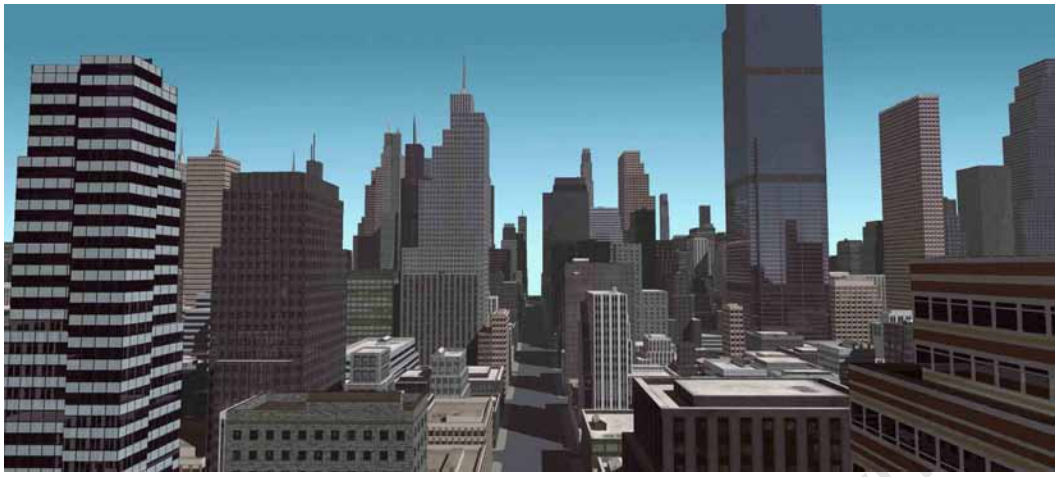


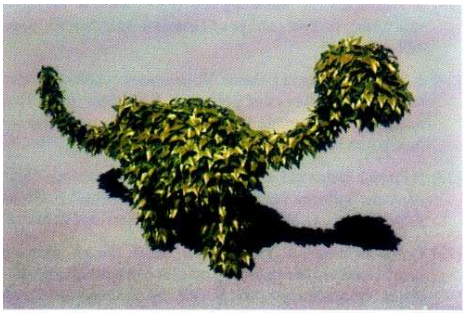
Figure 13: *An example of a city generated by the technique of Parish and Müller [2001].*

to the building design and relies on the previous stage having been completed. Following this reasoning, Muller et. al. assign priorities to the rules in their grammar, and in rewriting only consider rules of the lowest matching priority. This ensures that the derivation of a building model occurs in a controlled manner, from modelling the coarse shape down to the finer details, which are added at an appropriate later stage. Because the input to their system is a plot on which to construct the building, their grammar also allows for more complex constructs to be explored, such as the addition of trees, paths and other objects — giving rise to visually rich models, an example of which is shown in Figure 12.

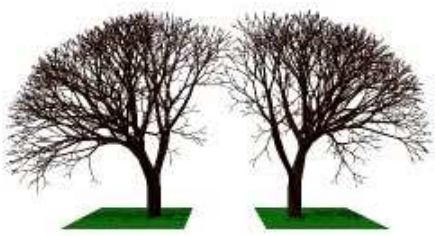
2.2.2 Trees and plants

In modelling the natural world, an important and complex component is the accurate modelling of trees and plants. Here, L-systems have been the primary technique used [Lindenmayer, 1968; Prusinkiewicz and Lindenmayer, 1990], largely inspired by the fact that L-systems mimic biological processes in their execution. Extensions to basic L-systems have also allowed for more interesting constructions. Prusinkiewicz et al. [1994], for example, demonstrate how an environmentally-sensitive L-system can be used to grow a plant to a fill a desired shape (see Figure 14(a)). Another extension, open L-systems, allows multiple objects to be generated simultaneously in a shared environment by providing a means for the L-system to query aspects of the environment such as spatial location, proximity of other objects or the amount of light reaching a point [Měch and Prusinkiewicz, 1996]. This provides for models in which plants compete for space, water and light (see Figure 14(b)).

Interactions between plants are not just limited to the use of extended L-systems, however. Deussen et al. [1998] tackle the task of modelling and rendering plant ecosystems, largely employing simulation techniques but also making use of image maps as a form of input. In this case, greyscale images are used to specify spatial distributions of plant densities and other plant characteristics such as their age and vigour. The authors also note that the use of image maps in this case allows for



(a) A topiary dinosaur [Prusinkiewicz et al., 1994]



(b) An example of two trees competing for light [Měch and Prusinkiewicz, 1996]

Figure 14: *Examples of some extensions to basic L-systems.*

certain interactions between plants to be captured, such as the effect of a tree crown on the vigour of undergrowth (by projecting the area of the crown onto the image map governing the undergrowth vigour and adjusting it accordingly).

In contrast to the synthesis of new trees and plants, another possibility is the use of photographs to capture a specific instance, as addressed by Shlyakhter et al. [2001] and Quan et al. [2006]. The key difference between these two techniques is in the preservation of structure of the object in question: Shlyakhter et al. [2001] restrict their approach to the modelling of foliaged trees — where their model may differ significantly in structure from the real instance yet appear visually the same — whilst Quan et al. [2006] seek to maintain the structure of the plant being modelled. Both techniques, however, make use of photographs taken from different positions around the tree or plant. The differences in these techniques arise from how these photographs are processed and how much photographic input is required.

Shlyakhter et al. [2001] make use of 4 to 15 photographs, for each of which the position of the camera is known, and which are segmented into the tree and background — thus producing the silhouettes of the tree as seen from the positions at which the photographs were taken. In order to infer a 3D shape, these silhouettes are projected to create *silhouette cones*, which are then intersected. This gives a bounding silhouette volume inside which the tree must lie. The branching structure of the tree is inferred from the silhouette volume by finding an approximation to the *medial axis* of the volume (see Ogniewicz and Kübler [1995]), which provides the trunk and first few levels of branches of the tree. An *open L-system* [Měch and Prusinkiewicz, 1996] is then used to expand on this skeleton, ensuring that the resulting tree shape tightly fits the silhouette volume and thus resembles the tree being modelled. The L-system also seeks to achieve a botanically plausible tree, by simulating the flow of photosynthates through the tree according to how much sunlight each leaf receives. Branches that require more energy to survive than is provided by their leaves are pruned from the tree.

In stark contrast, Quan et al. [2006] utilise between 30 and 45 photographs for their technique. However, they do not require that the camera position or parameters be known — these are calculated

along with a 3D point cloud, using a *structure from motion* technique (see Lhuillier [2005]). As each photograph is now registered with respect to the 3D point cloud, image gradient information can be associated with each point. This, coupled with both 3D and 2D distance metrics (the latter derived from the positions of 3D points in the 2D images), allows a graph to be constructed using k-nearest neighbour computations to define the edges. This graph then enables leaves to be segmented according to the solution of the two-label graph-cut problem. The authors are able to exploit the fact that all the leaves on a plant are very similar, by making use of a generic leaf model which is fitted to each segmented leaf. Once the leaves have been reconstructed, branches are reconstructed through user interaction and by modelling the branches as generalised cylinders, with the skeleton represented by a 3D spline curve.

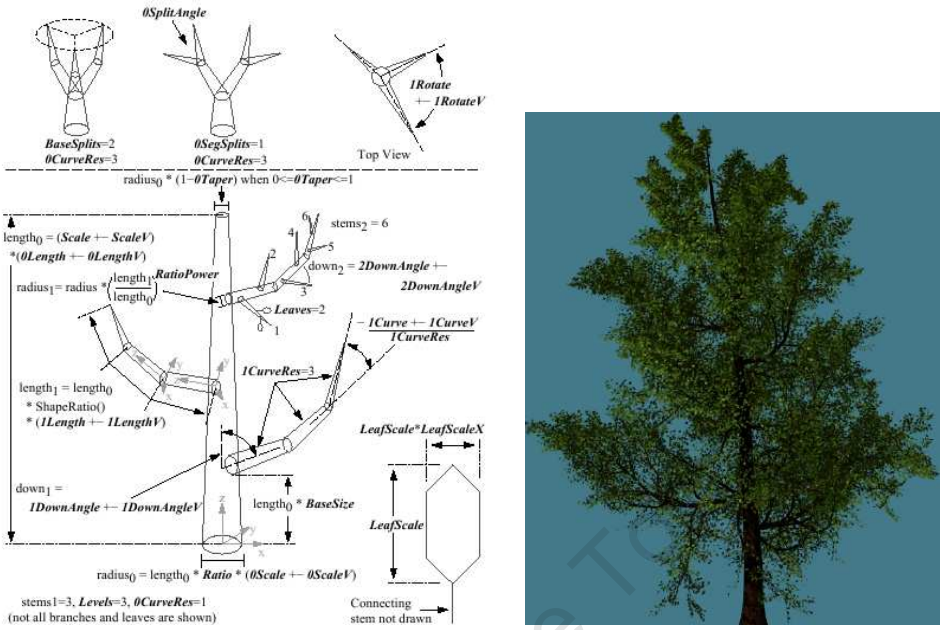
(a) A photograph of the plant being modelled (b) The recovered model in a new synthetic environment

Figure 15: *An example reconstruction from the technique of Quan et al. [2006].*

Even more possibilities exist for the procedural modelling of trees and plants. One such technique is that of Weber and Penn [1995], which aims to realistically model trees in an ad-hoc manner using 80 real-valued parameters. This wealth of information is used to describe tree characteristics such as shape, scale, branching variation, number of branching levels and angles of branching, to name but a few. Some of these parameters and their geometric significance can be seen in Figure 16(a), and an example of a tree generated with this system in Figure 16(b). Okabe and Igarashi [2003] went on to extend the work of Weber and Penn to a sketch-based system, which will be discussed in more detail in Section 2.3.

2.2.3 Terrain

Terrain forms an integral part of outdoor virtual environments, and based on the fact that landscapes and coastlines exhibit fractal-like properties [Mandelbrot, 1983] terrain is often modelled using fractals. Popular choices are fractional Brownian motion applied to Perlin noise [Perlin, 1985] and plasma noise [Miller, 1986], as these give good irregular noisy patterns similar to those found in nature. A more recent example of a technique that employs fractals for the real-time editing, synthesis and rendering of infinite procedural terrains is that of Schneider et al. [2006].



(a) An illustration of how parameters affect tree construction

(b) An example tree

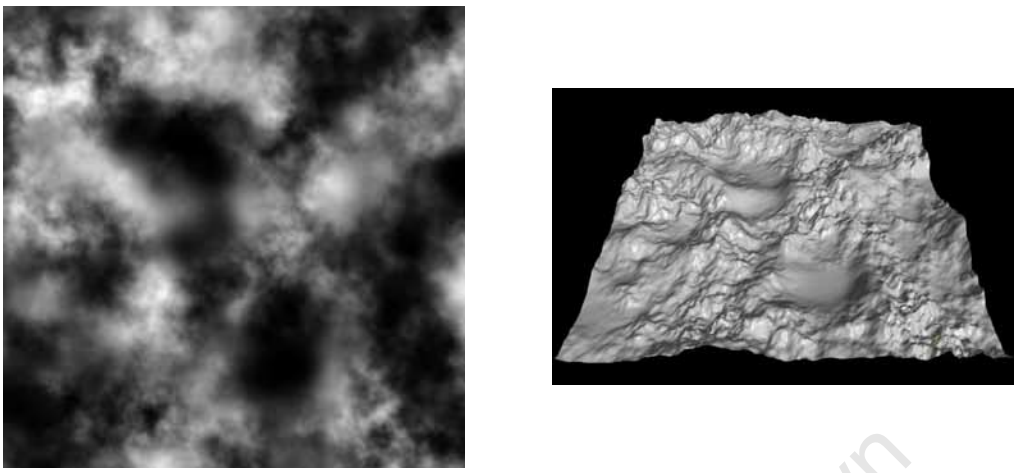
Figure 16: *Images from the work of Weber and Penn [1995].*

A chosen fractal technique is typically used to create a two-dimensional *heightmap*, which is then transformed into geometry by treating each value in the heightmap as a corresponding height (see for example Figure 17). Some applications require real-world topographical data: heightmaps in this context are more commonly referred to as *digital elevation models* (DEMs). There are several free sources of DEM data, the most complete of which is currently from the *Shuttle Radar Topography Mission* (SRTM) [Farr et al., 2007] and offers 3 arc-second (approximately 90m) resolution for most of the world.

Although fractals provide a good starting point for the synthesis of life-like terrain, they are by nature self-similar and so capturing topographical features involving multiple materials and varying rates of weathering is difficult. One approach, which has been used with some success, is to blend together several types of noise [Olsen, 2004]. Olsen observes that the cellular structure of Worley noise can be used to represent entire mountains, and that blending this with plasma noise gives a good mix of higher-level structural diversity as well as lower-level noise (Figure 18(a)). This does, however, result in unnaturally straight lines due to the nature of the Worley noise. Olsen addresses this issue by applying a perturbation filter [Ebert et al., 1994, pp. 90–91], which perturbs each co-ordinate of data a random magnitude along a random direction vector[2] (Figure 18(b)).

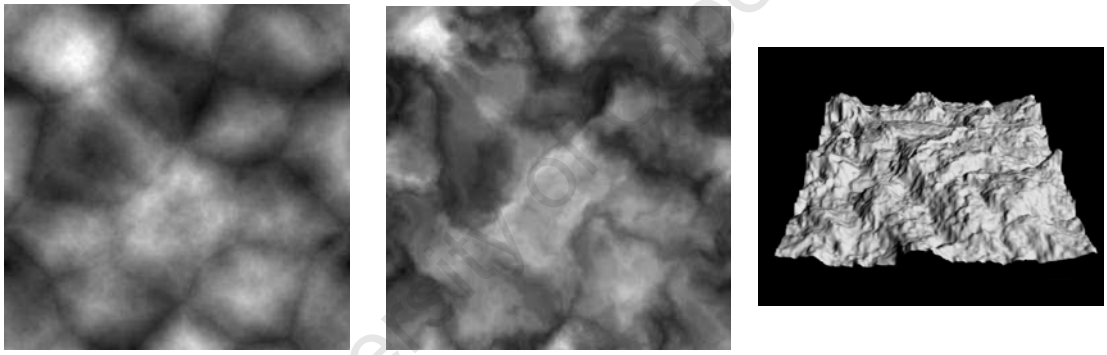
An alternative means for realistic terrain synthesis is to model the physical processes involved. Having used noise techniques to generated a reasonably plausible heightmap, Olsen goes on to simulate an erosion process on the terrain. Erosion techniques are broadly divided into two categories:

[2]Where random in this sense implies a continuous (i.e. noisy) randomness.



(a) (b)

Figure 17: *An example of a heightmap and its corresponding 3D terrain. Dark values in the heightmap indicate lower areas, and lighter values indicate higher areas.*



(a) (b) (c)

Figure 18: *(a) An example of a combination of Worley noise and plasma noise, (b) further application of a perturbation filter to (a) [Olsen, 2004], and (c) the 3D terrain corresponding to the heightmap in (b).*

thermal erosion and *hydraulic erosion*, both described by Musgrave et al. [1989]. Thermal erosion simulates pieces of material breaking loose and moving down slopes until friction overcomes the force of gravity, causing deposition. Hydraulic erosion, on the other hand, simulates water eroding or dissolving material and then depositing it according to water velocity and evaporation. These different classes of erosion give rise to quite different results — whilst thermal erosion wears away at the top of steep slopes to accumulate scree at their feet, hydraulic erosion creates drainage basins and carves deeper into existing valleys. In practise both are required for a truly realistic phyical simulation, but in some contexts only one or the other may be suitable — Olsen [2004], for example, focuses on creating procedural terrains for a multiplayer strategy game in which emphasis is placed on flatter areas of terrain, and so his approach places more emphasis on hydraulic erosion.

For a procedural terrain that looks truly realistic, one can also draw on real-world DEM data.

Zhou et al. [2007] use example DEM data and a user-supplied sketch of desired terrain features to synthesise new terrain (Figure 19). Their technique places importance on large-scale matching of features such as rivers and valleys between the user's sketch and the example DEM data, leaving all the finer details to be supplied by the DEM.

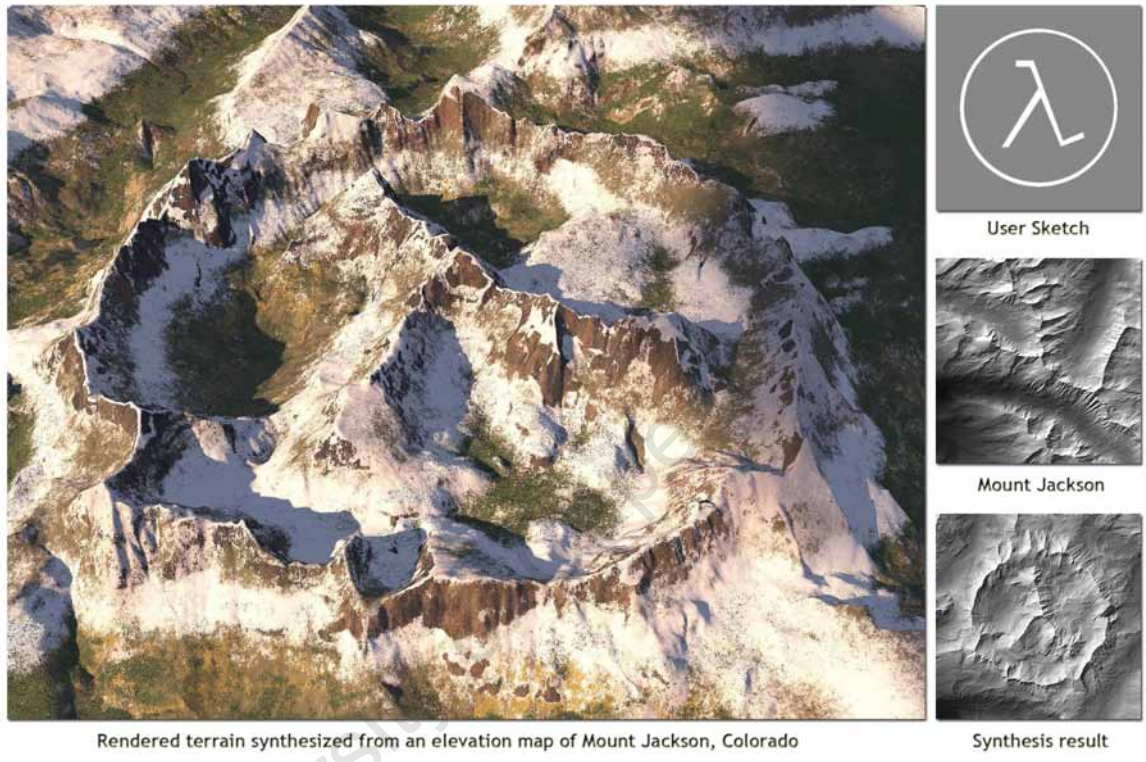


Figure 19: *An example of the terrain synthesis technique of Zhou et al. [2007], in which a user-supplied sketch and example DEM data are combined to synthesise new terrain.*

2.2.4 Clouds, sea and skylight

Parts of the environment which are often taken for granted are those things most distant — clouds, sea and sky — but without these an environment will tend to feel artificial.

Due to the fact that clouds and water are seldom static in real life, artists and designers typically aim for dynamic methods when creating these for use in a virtual environment. This, in itself, can be quite hard, as physically simulating cloud and fluid motion on a large scale is computationally expensive, and without special purpose hardware intractable for real-time animation. Additionally, David Ebert notes that understanding the mechanics behind physics parameters is often beyond many animators and modellers [Ebert et al., 1994, Chap. 9], and that abstracting these parameters away is imperative. As such, other methods that are not physically accurate but that still yield good results are generally used.

One of the simplest ways of achieving realistic renderings of water and clouds is using noise functions — with an added dimension for time, thus supporting animation. Early work [Perlin, 1985] modelled clouds using volumetric noise, and Ebert et al. [1994] gives a good overview of how fBm noise can be used to create a texture that gives the appearance of distant clouds. Ebert et al. [1994] also note that the cellular structure of Worley noise can be employed to create a bump-map for water that simulates a sea surface. (see Figure 20)

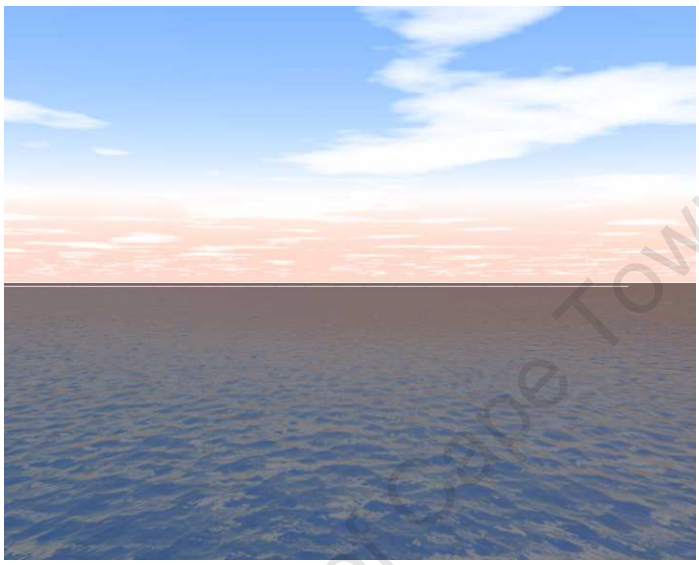


Figure 20: *An example of how noise functions can be used to capture clouds and sea in nature.*

Although noise is useful for the creation of distant clouds, it offers limited local control and does not offer the same level of realism as one approaches clouds or passes through clouds (for example in a flight simulator). Bouthors and Neyret [2004] model the shape of cumulus clouds through a hierarchy of quasi-spherical blobs, whose shape is defined by an implicit field that is influenced by neighbouring blobs. Although the authors only address the issue of realistic cloud modeling, they believe that their model is well suited for particle animation and advanced rendering techniques. Schpok et al. [2003] present a system for both modeling and rendering of realistic clouds, by representing clouds as groups of implicit ellipsoids that are rendered in slices. Low-level detail is achieved through the use of noise to define opacity, effectively subtracting away noisy regions from the surface of the implicit volumes that define the clouds.

Harris and Lastra [2001] and Harris [2003] address the issue of cloud modelling and rendering by representing clouds as point-sets which are pre-rendered into two-dimensional images and then interactively rendered as billboards, commonly referred to as *impostors*. As the user's viewpoint changes, impostors whose angle or distance have changed significantly relative to the viewpoint are re-rendered from the particle form, thus maintaining a reasonably accurate representation relative to the dynamic viewpoint. Harris additionally provides support for flying through clouds by splitting the cloud into multiple impostor layers. See Figure 21(a).

Wang [2004] extends the performance of Harris' technique by modelling the clouds as a set of cuboid

(a) (b)

Figure 21: *An example of the clouds generated by the techniques of (a) Harris [2003] and (b) Wang [2004].*

regions, each of which is rendered using a number of billboarded sprites that are sampled from 16 predefined textures. These are, in turn, also converted on-the-fly into impostors to further improve rendering performance. Wang also describes how her technique can be made dynamic by adjusting the transparency level of the rendered sprites, thus allowing for formation and dissipation of clouds. See Figure 21(b).

When modelling skylight, one of the simplest and most often used techniques is to simply use a predefined texture and map this onto a *skybox*. Often this texture is provided by a real-life panoramic photograph, and so the results are extremely realistic but static: to capture dynamic and realistic skylight typically requires physical simulation. One such technique is that of Preetham et al. [1999], in which analytical formulae for the calculation of daylight are derived by fitting physical models to large datasets of real-world measurements. Parametrically the technique is very easy to use, with intuitive inputs such as the geographical coordinates (longitude and latitude) of the viewpoint, the current date and time, and the *turbidity*[3] of the environment. With these parameters, the physically-based model is used to give accurate sky and ambient light colouring, which can then be utilised for realistic effects in other parts of the environment — such as accurate lighting of clouds and water, for example.

Figure 22: *An example of daylight as presented in Preetham et al. [1999].*

2.2.5 Music and speech

Whilst the focus of this chapter has been primarily on procedural modelling related to computer graphics, this is not to say that it is not applicable to other domains. Another area which has seen much research is the creation of sound, especially music and speech. With regard specifically to

[3]Turbidity is a parameter used in atmospheric modelling to describe haze, and represents a measure of the fraction of atmospheric scattering due to visible particulate matter as opposed to gaseous molecules.

music, this is generally referred to as *algorithmic composition*. For example, Essl [1995] describes a semi-automatic technique in which a user "performs" a piece by interactively activating or deactivating 24 musical structure generators. These generators pseudo-randomly create melodies subject to some controlling parameters, and are constructed in such a way that each shares some musical structure with another generator — thus allowing combinations of the randomly generated music to blend harmoniously when listened to. Other algorithmic composition techniques include the use of fractals [Thompson, 1999] and L-systems [Prusinkiewicz, 1986].

When considering speech synthesis, much of the efforts have focused on establishing datasets of neutral *diphones* and then synthesising new speech by a combination of the diphones in the database (such as the MBROLA project [Dutoit et al., 1996]). Another avenue which has been approached is in the synthesis of emotional speech: Oudeyer [2003] describes a procedural method for the creation of sounds that convey emotion.

2.3 Interfaces

As has been demonstrated, procedural modelling techniques offer a wealth of possibilities for the creation of complex objects and environments. They abstract away the complexity of the final object, seeking to model that complexity with a smaller set of parameters. Many of the software applications available for modelling make use of some procedural techniques, but additionally give the user very low-level control — in the case of computer graphics, they allow for direct manipulation of the 3D polygon-mesh comprising a model. Alterations can be made to individual vertices and faces of the model or to groups of these, allowing the user to stretch, transform, add to or subtract from the model as desired.

However, the low-level of control offered by these packages is often prohibitive to novice or less technically able users, and obtaining realistic output from the procedural techniques available requires a substantial amount of work and adjustment of procedural parameters. Due to the low-level focus, these applications are not typically thought of as interfaces to procedural modelling, since the techniques used are typically quite simple and the interface to the procedural techniques is through the direct manipulation of procedural parameters.

With these shortcomings and the fact that procedural models are so widely applicable, interfaces to these models are becoming increasingly important and worthy of some consideration.

2.3.1 Freehand sketching

Many complex objects exhibit shapes that are not easily captured via mechanical rules or inferences. Human designers, however, are far more adept at capturing such shapes through the use of sketching, and these sketches can be used to guide particular procedural modelling techniques.

One of the earliest interfaces for sketching 3D shapes is the gestural interface known as SKETCH [Zeleznik et al., 1996]. This was developed primarily for the creation and editing of rectilinear objects, and made use of the geometric attributes of gestures to infer numerical parameters defining the objects. For example, by drawing three lines that meet at a point the user is able to create a cuboid whose position and size are determined by the junction point and lengths of the lines. Many other primitives are available in the SKETCH system, including volumes of revolution, cones, cylinders, extrusions and superquadrics.

In 3D modelling, a common use of freehand sketching is to infer 3D geometry or shape from the 2D sketch by considering the 2D sketch as a silhouette. Teddy [Igarashi et al., 1999] is one such technique and it allows for the easy creation of stuffed animals and similarly rotund objects. 3D shape is inferred from the 2D silhouette by inflating the region surrounded by the silhouette — wide areas are taken to be fatter and are inflated more, whilst narrow areas are considered thin and inflated less. Additional operations are also provided via the sketch interface that allow the model to be extruded, to be cut, to have strokes painted onto the 3D surface, and to have either painting strokes or 3D geometry removed by "scribbling" on the model. Furthermore, these operations are all interactive in the sense that after each stroke is drawn by the user, the 3D model is updated. The inferred 3D model can also be rotated at any time, allowing the user to make use of the sketching interface from any angle.

Figure 23: *A tree created using the technique of Okabe and Igarashi [2003]*

SmoothSketch [Karpenko and Hughes, 2006] builds on the work of Teddy, by allowing for a user's sketch to be more complex than the simple closed curve imposed by Teddy. The use of cusps and T-junctions in the sketch allows for hidden parts of the contour to be visually depicted, and SmoothSketch is able to infer a smooth solid shape that matches these visible contours. Additionally, SmoothSketch allows objects that are topologically more complex than the restrictive spherical topology imposed by Teddy.

Another technique that infers 3D geometry from 2D sketching is that of Okabe and Igarashi [2003], which describes a sketch-based solution to the 3D modelling of trees — a task which is typically addressed using L-systems. One of the chief arguments in favour of sketch-based systems is that they cater for novice users, unlike L-systems which are geared more towards expert users. The driving force behind their technique is that they infer 3D tree geometry from a 2D sketch, based on the assumption that trees spread their branches uniformly. From an initial silhouette sketch of the profile of a tree, the trunk and branches are identified and the branches are distributed uniformly around the trunk. Their system then recognises several strokes that allow the model to be edited: branches can be added and removed, large groups of new branches can be added according to prediction patterns derived from the work of Weber and Penn [1995], branching styles can be propagated across the tree, and leaves in the form of planar polygons can automatically be added

to terminal branches.

In Ijiri et al. [2005], the authors address the related problem of modelling flowers and *inflorescences*[4]. Here, sketches are used to describe the shape of the *floral receptacle*, or flower petals, and of the inflorescence's central axis. A sketched curve drawn by the user is used as the outline for a volume of revolution defining the floral receptacle, whilst the 2D outline gives the shape of a petal and an additional stroke is used to indicate the center vein of the petal. Further strokes can also be used to deform flower petals on both a local and global scale.

Whereas the techniques discussed thus far have all aimed at creating specific instances of objects, the work of Baxter and Anjyo [2006] makes use of sketching interfaces for the purpose of *unique instancing* — the generation of new content that is similar but not identical to a set of input examples. By taking as input from the user a set of simple line drawings, or "doodles", they establish stroke correspondences between the drawings and make use of latent variable methods such as a Gaussian Process Latent Variable Model (GPLVM) [Lawrence, 2004] to extract a low-dimensional space — a *latent doodle space* — representing the drawings. Sampling of the latent doodle space then allows for the generation of new drawings that are similar, but not identical to, the input drawings. The ramifications of such a technique are significant — in nature, instances of the same type of object look visibly similar but exhibit small differences, and so being able to capture this dynamic automatically, and without needing to adjust every instance by hand, is very important. One application presented by the authors is the synthesis of handwriting — every person has their own style of handwriting, but, nevertheless, every time we write a particular letter it will be slightly different.

2.3.2 Interfaces for parametrised procedural models

In the case of numerically parametrised models, the number of parameters used by a technique can be unmanagably large (for example in the work of Weber and Penn [1995], who require 80 parameters for a single tree). The correlation between these parameters and the final object is also often not very intuitive for a less technically able user. Additionally, parameters may have complex interactions — particularly in an environment where the various procedural elements are tightly coupled, such as in an outdoor landscape — and the nature of these interactions may be too complex for a user to fully grasp. It cannot be denied that procedural models offer powerful procedures for the creation of complex objects, and it would be advantageous to keep these benefits whilst at the same time providing a better interface that eliminates the complexities introduced by procedural parameters.

Marks et al. [1997] propose a *design gallery* interface to parameter setting, in which the parameter space can be visually explored by browsing through a gallery of selected outputs. To create the gallery, a random set of *input vectors* in parameter space is chosen and used to generate a corresponding set of *output vectors*. The output vectors need not be the actual output from the technique, but should be representative of the output, and differences in the output vectors should correspond

[4] An inflorescence is a branch with multiple flowers

to differences in the procedural content generated. For example, if the output is a rendered image, then a suitable output vector might be a thumbnail of that image; if the output is an animated model, then a suitable output vector might be some low-level statistics pertaining to the animation, such as the average speed and position of key points such as joints or limb extremities. Once the output vector associated with each input vector has been calculated, the pairs can be hierarchically organised according to the similarities of their output vectors, and they can also be spatially arranged in two dimensions by applying *multi-dimensional scaling* [Cox and Cox, 2000].

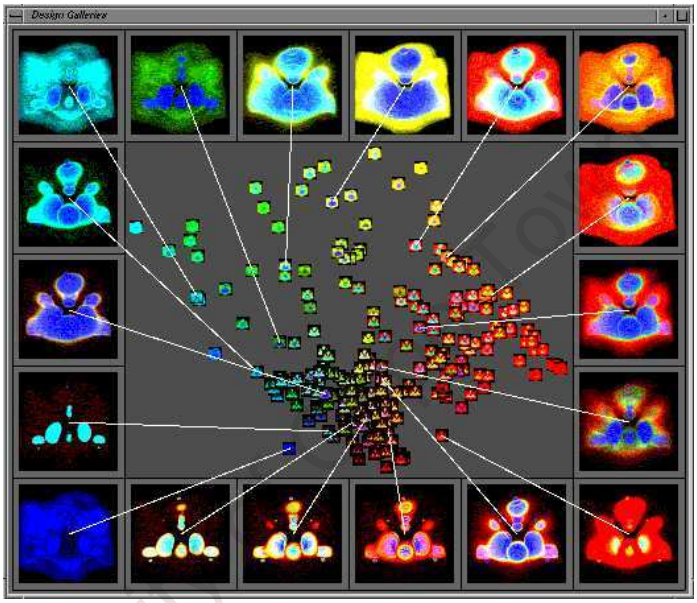


Figure 24: *An example of a design gallery from the work of Marks et al. [1997], demonstrating how the technique can be used to choose suitable opacity and colour transfer functions for volume rendering.*

Another possible interface to parameter setting involves the use of *genetic algorithms* [Holland, 1975, 1995]. These are processes that model the evolution of a population of entities, through interactions that closely follow biological reproduction patterns — discussed in much greater detail in Chapter 5. In applying genetic algorithms to the creation of content by procedural means, we first need to codify the concept of the possible population of content and what constitutes an individual in that population. Fortunately, the nature of procedural modelling techniques make this quite easy — each technique takes some input, and generates corresponding output, so provided that the techniques are deterministic then a given set of input parameters will *always* produce the same output[5]. Now an individual of our population is simply any instance of valid parameter values, and the possible population is the set of all possible individuals (in other words, the set of all input parameter value combinations). A well-known example of the use of genetic algorithms is in the creation of *biomorphs* [Dawkins, 1996], and Bedwell and Ebert [1999] provide an example of applying genetic algorithms to the modelling of algebraic surfaces.

[5] And if one really wanted a non-deterministic technique to be admitted, then this can be done by adding an extra procedural parameter which is used to seed the random number generator.

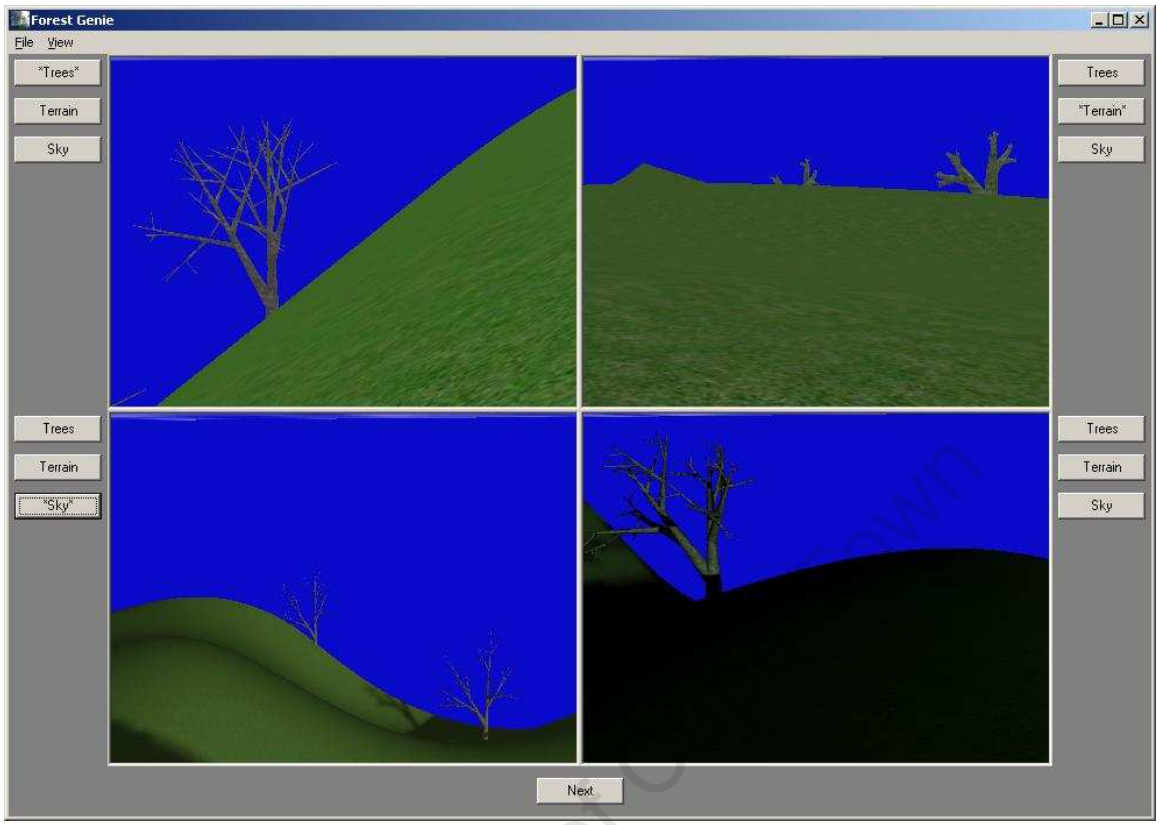


Figure 25: *The interface to genetic algorithms for procedural environment generation of Merry et al. [2003].*

What remains is to define a *fitness function*, or a means of evaluating how close individuals in our population are to the content that we are aiming to generate. Merry et al. [2003] investigate the use of user feedback to establish fitness, by showing the user several virtual environments and asking them to indicate which of the environments is closest to the goal on several criteria. These ratings are used to update a probability vector, in accordance with the *population-based incremental learning* algorithm described by Baluja [1994], which is used to generate new individuals for the following iteration. They tested their system through user experimentation in which users were asked to produce a specific set of environments. Users were required to use both the genetic algorithm approach as well as an interface where the procedural parameters were directly manipulated, with some users using the genetic algorithm interface first and others the direct manipulation interface first to avoid learning bias. Their results showed that the direct manipulation interface performed better for all criteria measured — namely the difficulty of the task, how close they felt their result was to the target environment, and which interface the users' preferred. Given the success of other interactive evolutionary techniques in computer graphics [Sims, 1991; Rowland and Biocca, 2000] the authors surmised that their implementation was flawed, rather than the concept. They suggest that the speed of their environment generator needs to be improved, so as to meet the requirements for interactive evolution set out by Sims [1991]. In addition, they note that the number of parameters used in their experiments may have been too low, making it possible for users to easily learn how

individual parameters affected the generated environment — their system made use of 21 parameters or 70 bits, in stark contrast to the 104 degrees of freedom and 1625 bits used by Rowland and Biocca [2000] or the unbounded genetic space employed by Sims [1991].

2.4 Summary

In this chapter, we have explored the realm of procedural modelling, focusing on the core techniques that are used to create the models, considering common areas of application, and finally examining higher-level interfaces. We have seen how simple models can give rise to complex and detailed output, and how procedural models assist in the creation of content by having the computer assist with the computationally intensive and repetetive tasks. In the next chapter, we build on this knowledge by considering adjectival interfaces, and how these might be employed to provide an intuitive interface, yet still effectively utilise the power of procedural models.

Chapter 3

Overview of adjectival technique

As was discussed in Chapters 1 and 2, procedural methods provide effective means for quickly creating large quantities of complex digital content, but they are limited in that their parametrised interface can inhibit effective usage. Whilst some of these methods can be simplified by reducing the number of parameters, this in turn reduces the variety and complexity of obtainable output. As motivated in Chapter 1, what is needed is an improved interface to procedural methods that does not sacrifice the flexibility that they afford.

3.1 Motivation for adjectives, and related work

In Chapter 2, the notion of *chunking* — combining together simple objects into more complex representations [Hofstadter, 1979] — was a recurring theme, and is in fact central to the notion of almost all digital content. Consider a virtual environment which is, in effect, nothing more than a single chunk of information composited of smaller components that are combined or connected in various ways. We can reason about the environment as a whole, on a high level, or we can break it into its components and reason about those at some lower level. In Chapter 2, we also saw a trend to develop techniques which operate at higher levels — with this in mind, we approach the subject of creating digital content in a top-down fashion by first reasoning about the entire content as a complete unit.

Again with reference to virtual environments: suppose a user wishes to create a VE. They have a mental picture of what this VE looks like, but how can they convey this to the computer? Since our aim is to provide an interface that is accessible to non-technical users, a better approach would be to leave the computer out of this question for the moment, and simply phrase it as "how can they convey this to another person?".

When one person wants to convey an idea to another person, the natural way that they do this

is to *describe* important features, so that the other person can build a mental picture. The use of verbal or textual descriptions in tagging media for later synthetic constructions has been explored extensively [Joshi et al., 2006; Rose et al., 1998; Polichroniadis, 2001], as has the use of prepositions to relate objects in the synthesis of environments from text [Coyne and Sproat, 2001]. This wealth of research indicates the importance of such natural language interfaces, and we discuss several of these in section 3.1.2. The act of describing, and in particular of describing specific details, involves interspersing the description with *adjectives* that qualify certain aspects of the description.

With this in mind, we propose an interface in which the user can choose from a number of adjectives that describe the procedural content they wish to create. Such an interface should be familiar to the user, as it mimics the way in which they interact with other people.

3.1.1 Related work in the use of adjectives

First, it is worth noting that typical procedural modelling interfaces make use of adjectives in their labelling of the various controls available to the user. We will not consider these, as the adjectives do not so much *define* the interface as provide a convenient labelling for the parameters. Instead, we examine techniques where adjectives are used as an active stylising tool for computer graphics, which is how we intend to utilise them.

The major area in which the use of adjectival descriptors has been explored is in the creation of stylised character motion [Polichroniadis, 2001; Rose et al., 1998; Unuma et al., 1995; Brand and Hertzmann, 2000]. A goal of stylised character motion is to produce motion from a given action (such as "run"), and some additional descriptors of how the action is done (such as "run quickly"). The focus of stylised motion is on the additional descriptors, which indicate perturbations to the neutral character movement associated with the relevant action. Through previous research, a common paradigm for representing these stylistic features has emerged. The available descriptors are all represented by adverbs, each of which defines an axis in a multidimensional space referred to as *adverb space*. The stylisation of the motion is then simply described by providing a vector representing a single point in adverb space. Given a point in adverb space and an action, the problem is then to produce a corresponding motion. This has been approached in a number of ways.

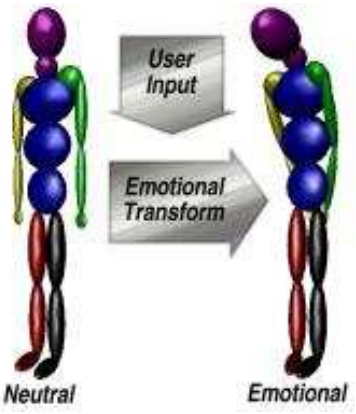


Figure 26: *A graphical summary of the technique of Polichroniadis [2001].*

Rose et al. [1998] address the problem by initially labelling each of a set of example motions for a given action with adverb values, thus placing each example motion in the adverb space for that action. Given an arbitrary point in adverb space, the corresponding motion is then found by interpolating

between the example motions using a combination of radial basis functions and linear polynomials. The linear polynomials provide an overall approximation as well as extrapolating outside the convex hull defined by the example motions, whilst the radial basis functions locally adjust the linear approximation to better interpolate between the actual example motions. Due to the continuity imposed by radial basis functions and linear polynomials, and the way in which the authors ensure correspondence between the various example motions, continuous motion with varying styles can be achieved by continuously adjusting the point in adverb space. The authors go on to describe how a verb graph can be constructed to allow morphing between different actions, adding another level of sophistication and realism to the system.

Brand and Hertzmann [2000] also employ an interpolation-based technique, albeit more complex in nature than that of Rose et al.. They extend *hidden Markov models* (HMM) to take on an additional multidimensional style variable, giving *stylistic hidden Markov models* (SHMMs), which define an entire space of HMMs — fixing the style variable gives a unique HMM from which motion can be produced by transitioning around the states. Such a formulation is automatic, in the sense that the number of styles inherent in the training data and the association of these styles with various parts of the training data can be inferred automatically. The resulting style variable is similar to the adverb vector of Rose et al. in that each dimension references a particular style exhibited by the data, and thus any combination of styles can be specified to produce appropriate stylistic motion.

Unuma et al. [1995] attack the problem by first representing periodic character animation as a Fourier series expansion. With this representation, they show how interpolation between two motions can be achieved by interpolating the Fourier coefficients, and also how a feature of one motion can be extracted by noting the difference in Fourier coefficients of two motions. By adding an interpolant of these differences to the Fourier coefficients of a different motion, they show how the feature embodied in the difference coefficients can be imparted to the motion. For example, the notion of briskness could be extracted by noting the difference between a brisk walk and a normal walk, and then applied to a running motion to give a brisk run. Unlike the techniques of Rose et al. and Brand and Hertzmann, synthesis of new motion is restricted to the axes of adverb space — as the authors only show how one style may be used to augment another motion.

Like Unuma et al., Polichroniadis [2001] seeks to represent styles independent from motion — in contrast to Brand and Hertzmann and Rose et al., who interpolate between given motions to achieve style. Polichroniadis' technique is based on the application of a transform to a given motion, and capturing style within the transform. By sampling the available transform space, applying these transforms to a neutral motion and then allowing a user to choose which styles are characterised in each transformed motion, a library of transforms and their associated styles is built up. To then synthesise a new stylised motion, the user specifies the style and this is used as a lookup into the library to choose a number of possible matching transforms. These are applied to the neutral motion being stylised, and the user can pick which stylised motion best represents the effect they seek. The chosen motion can then be iteratively used to repeat the process and refine the library search, in a similar form to the operation of a genetic algorithm. Polichroniadis showed the effectiveness of

this technique by learning a transform library from a walking motion, and then applying it to other forms of motion such as sitting down, waving, drinking water, placing objects in an environment and picking up objects — with user tests showing that the styles learnt from the walking had been transferred successfully to other actions.

Since 2001, there has been much work in the field of stylised character motion, but the focus has shifted away from the use of adjectival descriptors. Most new techniques have instead targeted the problem addressed by Unuma et al. [1995] — capturing the style inherent in one form of motion and applying it to another form of motion. For example, Grochow et al. [2004] estimate a *probability distribution function* (PDF) for a given motion, which describes the likelihood, over the set of all possible poses, of a given pose belonging to the motion. They show how multiple PDFs from different styles can be combined into a new PDF, from which motion can be synthesised. Other notable techniques are those of Hsu et al. [2005] and Liu et al. [2005]; we omit the details here as they do not relate to adjectival descriptions.

3.1.2 Related work on more general natural language interfaces

Natural language interfaces, as introduced earlier in this chapter, have been used extensively in the past to support a number of techniques. These span a broad spectrum of applications, most notably efforts to enable a computer to hold a conversation with a human being (largely inspired by Weizenbaum's ELIZA [Weizenbaum, 1966]). We are, however, primarily interested in the use of such techniques as applied to procedural modelling and computer graphics, and so will focus on a few examples of work in these specific fields.

Natural language has been used in conjunction with computer graphics in two major areas: in describing the relationships between objects [Kahn, 1979; Coyne and Sproat, 2001; Yanai and Okada, 2006] and in the tagging or labelling of objects [Barnard and Forsyth, 2001; Joshi et al., 2006].

Kahn [1979] represents some of the earliest work available, and describes the synthesis of animation from a high-level description of a film. The system is able to extract information about characters, relationships, scenes and global film descriptors (such as variety, complexity and the length of film) from a textual description. Most of this information is inferred from a number of pre-chosen nouns, verbs, adjectives and adverbs, each of which is represented by what is referred to as a *computational actor*. Each computational actor is responsible for the adjustment of parameters used in the synthesis of the animation, and is programmed to specifically adjust them so as to reflect the concept captured by the actor. Consequently, these concepts are all from the perspective of the programmer, and so if another user has a different perception of the meaning of a particular concept then the results are likely to conflict with their expectations. Relationships between these computational actors are also specified (for example, to indicate opposite and similar meanings), which allows for comparisons and for indirect suggestions of appropriate parameter values. One of the most important features of this work is the inference of relationships between characters, and how the computational

actors reponsible for these relationships ensure that the parameters governing the characters are in similarly appropriate relationships to each other. Unfortunately, the reliance on a predefined set of computational actors means that only animations which use those concepts can be synthesised, and, as has been mentioned, the portrayal of these concepts accords with the programmer's perceptions, and not necessarily as the director might intend.

Related to this is the work of Badler et al. [2000], who describe a method for instructing virtual agents to carry out actions specified through natural language. The authors represent each action by a semantic tree, where each node contains all the information necessary to characterise a given action, as well as links to constituent actions or actions that must be performed before, after, or concurrently with this action. A number of these nodes are pre-specified in an abstract format, omitting details of the actual agents or physical objects involved, and comprise a dictionary of available actions in the system. Users are then able to specify actions in realtime using natural language sentences. These are parsed into various parts of speech, and these are then matched against the dictionary of available actions to choose a suitable action for the agents to carry out. The main limitation here is in the populating of the dictionary — every possible action that an agent might perform must be described, and these actions are quite specific to the environment in which the agents are acting.

WordsEye [Coyne and Sproat, 2001] is an automatic text-to-scene conversion system which draws on a variety of smaller tools in order to generate a three-dimensional scene. Given a piece of text, a lexical analysis is performed to determine a dependency hierarchy. In particular, this provides information on what objects are being described, which groups of objects are related by prepositions, and what adjectives and adverbs apply to which nouns and verbs. The object names are used to index a database of models, which have additional tags (such as colour, shape, hierarchy of components or parts, and skeletal pose information), to better match models to objects in cases where additional descriptors are attached to the objects in the dependency tree. Several spatial relationships are then recognised to correctly place the objects in the scene relative to each other. For objects with skeletal pose information, additional inverse kinematic constraints are applied to ensure that the actions being performed by characters look realistic. Finally, any modifying attributes such as colour, size or shape are applied to the objects. The system has some implicit common-sense constraints built-in, as well as a means for dealing with conflicting constraints. The major limiting factor of the system is the database of models available, although the user is free to expand this by adding their own models and associated tags.

Yanai and Okada [2006] attack a different problem, namely the modelling of objects comprising simple geometric shapes. They describe a system that accepts natural language as input, and interprets this as a sequence of instructions for the step-by-step construction of a scene. Each instruction represents a deformation of the scene, which can involve adding a new object, modifying the attributes of an existing object, or relating groups of objects through CSG operations such as intersection, union and difference (for an example, see Figure 28). Their system also supports the animation of objects through affine transformations, which are also parsed from the natural

Figure 27: *An example of a scene generated by WordsEye [Coyne and Sproat, 2001], which was described by the following text.* "John uses the crossbow. He rides the horse by the store. The store is under the large willow. The small allosaurus is in front of the horse. The dinosaur faces John. A gigantic teacup is in front of the store. The dinosaur is in front of the horse. The gigantic mushroom is in the teacup. The castle is to the right of the store."

language.

PLEASE INSTRUCT-> Put a sphere.
PLEASE INSTRUCT-> Paint the sphere white.
PLEASE INSTRUCT-> Make the sphere 1.5 times.
PLEASE INSTRUCT-> Move the sphere 2 behind.
PLEASE INSTRUCT-> Name the sphere body.
PLEASE INSTRUCT-> Put a sphere.
PLEASE INSTRUCT-> Move the sphere 2 down.
PLEASE INSTRUCT-> Move the sphere 2 behind.
PLEASE INSTRUCT-> Make the sphere 0.7 times.
PLEASE INSTRUCT-> Make the sphere 1.7 times.
PLEASE INSTRUCT-> Make the sphere 0.5 times.
PLEASE INSTRUCT-> Make the sphere big.
⋮

(a) A list of instructions to the system

(b) The resulting object — a snowman

Figure 28: *An example of the technique of Yanai and Okada [2006], illustrating the construction of a snowman.*

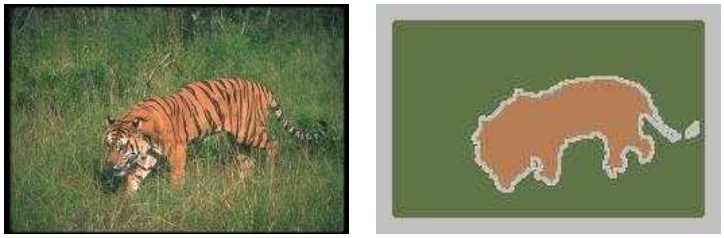
Using verbal descriptors in a different context, Barnard and Forsyth [2001] present a method for organising a dataset of images by combining the semantic information given in text associated with each picture, with visual information given by features extracted from the images. They organise the database into a hierarchy in which the leaf nodes are small clusters of similar images, and progressively higher levels in the hierarchy abstract out progressively more general words and image features present lower in the hierarchy. This hierarchy allows for collections exhibiting more general concepts to be viewed, by sampling from the clusters that lie below that concept in the hierarchy, and by traversing up the hierarchy one can determine the concepts describing each image in the dataset. An important facility for image databases is the ability to search the database, and the authors describe how the hierarchy allows them to compute, for each image, a set of probabilities that the image is associated with each of the search terms. Such a formulation allows for pairs

of terms that may never appear together in the textual descriptions of an image to still be linked through visual similarities with other images (see Figure 29). Their searching formulation has the advantage that it can be used to automatically illustrate text by selecting images which emit high probabilities with respect to the text. The authors then go on to show how the opposite can also be achieved with their system — namely the automatic annotation of external images with suitable textual descriptors — which has significant implications, particularly for extending the search of a database to include a dynamic set of images not in the database (see Figure 30).

TIGER CAT WATER GRASS TIGER CAT WATER GRASS TIGER CAT GRASS TREES

TIGER CAT WATER GRASS TIGER CAT GRASS FOREST TIGER CAT WATER GRASS

Figure 29: *An example from the technique of Barnard and Forsyth [2001], showing images that were retrieved from a database when searching for the keywords "river" and "tiger". The text below each image indicates what keywords the image was tagged with — note that none of the images have the keyword "river", but the system infers a connection via the keyword "water" as many other images in the database were tagged with both these keywords.*



(a) (b)

Figure 30: *An example from the technique of Barnard and Forsyth [2001] illustrating automatic annotation of images. The photograph in (a) is segmented into the blobby representation in (b), and by comparing this to images in the database the system can infer likely tags for the photograph. In this case, the system predicted the following tags (in order of decreasing probability): tiger, cat, grass, people, water, bengal, buildings, ocean, forest, reef*

Similarly to Barnard and Forsyth, Joshi et al. [2006] make use of textual annotations associated with images to automatically illustrate a story. The key differences between the two techniques are

in the format of the textual annotations used, and in the ranking system of which images best match the text being illustrated. Where Barnard and Forsyth use only simple individual words as tags for their technique, Joshi et al. allow much more natural annotations in the form of complete sentences and even paragraphs: they make use of WordNet [Fellbaum, 1998] to establish relationships between synonyms, meronyms and hypernyms[1]. To rank the images that best match a story, a mutual reinforcement process is solved, in which it is determined how much every image contributes towards the rank of all other images, based on their visual and lexical similarities to components present in the story being annotated.

YouTube and Digg are well known examples of related interfaces that employ tagging to facilitate convenient searching and filtering of digital content.

3.2 Principle of the technique

Having identified adjectival descriptions as one possible means for addressing the problems with existing interfaces to parametrised procedural models, what remains is to concretely link these two concepts. That is, to find some way in which the adjectives provided by the user can drive the procedural model generation, by mapping to correct procedural parameter values.

Suppose that the user can choose adjectives from a set $\boldsymbol{A}$ to describe their desired content. In the same style as the adverb space employed for stylised character motion, we define *adjective space* $\mathcal{A}$ to be a subset of $|\boldsymbol{A}|$-dimensional real values, $\mathcal{A} = [-1; 1]^{|\boldsymbol{A}|}$ (where $|\boldsymbol{A}|$ denotes the number of adjectives in the set $\boldsymbol{A}$). Each dimension in $\mathcal{A}$ thus relates to a unique adjective in $\boldsymbol{A}$. The value x in any dimension of $\mathcal{A}$ is a real value in the range $[-1; 1]$ and is the *scalar value* associated with the relevant adjective. The scalar value allows for the impact of the adjective to either be reduced or enhanced, with a value of -1 indicating a complete absence of the adjective and 1 indicating absolute presence of the adjective. This allows the user to quantify abstract concepts such as the use of the word "very". Note that the choice of $[-1; 1]$ is reasonably arbitrary as any closed range could be scaled and translated to this interval if so desired. The use of a range whose midpoint is 0 is, however, useful and will be discussed in more depth in Chapter 5[2]. One could also consider using the value -1 to indicate the opposite meaning of an adjective, where suitable, but in our system we prefer to capture such relationships between adjectives through the use of the WordNet database — this will be discussed at greater length in Section 3.2.1.

With the above formulation of adjective space, a point in $\mathcal{A}$ describes a specific set of scalar values associated with the adjectives in $\boldsymbol{A}$, and $\mathcal{A}$ represents the set of all possible descriptions that a user could make.

[1] A meronymic relationship indicates that one word is the component part of another, for example *beak* is a meronym of *bird*; whilst a hypernymic relationship indicates that one word is a more specific type of another word, for example *animal* it a hypernym of *dog*

[2] To briefly summarise, this allows for function approximation techniques to be unbiased in areas of the domain for which there is no training data.

Similarly, suppose that $\boldsymbol{P}$ represents the set of parameters controlling the procedural model. Without loss of generality, assume that each parameter is represented by a single, scalar, real-valued variable. This assumption is well founded since we can establish mappings for other parameter types (such as boolean, integer and enumerant parameters) to and from real values[3]. We then define *parameter space*, $\mathcal{P}$, to be the $|\boldsymbol{P}|$-dimensional reals, that is $\mathcal{P} = \mathbb{R}^{|\boldsymbol{P}|}$. Hence a point in $\mathcal{P}$ describes a specific set of real values associated with the parameters in $\boldsymbol{P}$, and $\mathcal{P}$ represents the set of all possible outputs that could be generated by the procedural model.

Recall that our aim was to establish a means for mapping the user's description into procedural parameters. With the definitions of adjective space and parameter space provided, this is mathematically expressed as the need to find a function

$$f : \mathcal{A} \rightarrow \mathcal{P}$$

Determining f is a non-trivial task; Chapter 4 discusses various approaches for doing this, which we will refer to as *function approximation techniques*. Moreover, different users will express themselves differently, meaning that if f accurately models one user's descriptions it will be unlikely to precisely model the descriptions of another user. There are several means for dealing with this problem, which include the following:

- **Determine f on a per-user basis.** Each user of the system needs to train the system to generate a unique f.
- **"Artist's vision": f is pre-determined.** Since the generation of digital content could be considered to be an art-form, one could take the view that individual users' opinions may differ but it is the artist's opinion that counts. To this end, one or more experienced users could train the system beforehand to generate a single f function, and then all successive users would make use of this function.
- **Synthesis of opinion.** The reason that humans are able to communicate effectively with adjectival descriptions is that we have some common knowledge or understanding. By collecting small quantities of data from a large number of users, we could combine all of these to obtain a function that could potentially model the consensus of the population as a whole.
- **f is mostly pre-determined, subject to small alterations on a per-user basis.** A slightly less radical approach is to suggest that, although the artist's opinion is important, individual users are allowed to interpret the virtual environment in slightly different ways. This would result in a function $f = h(g)$, where g is a function trained as the "artist's vision", and h is determined on a per-user basis through a smaller training set.

[3]For example, to convert from a real to an integer value, one could simply round the real value to the nearest integer. To convert a real into a boolean, one could divide the reals into two subsets and say that a real value less than 0 corresponds to the boolean value *false*, whilst a real value greater than or equal to 0 corresponds to the boolean value *true*.

Exactly which technique would work best is largely dependent on the characteristics of the underlying function approximation and the quantity of data required to determine the function, and so we defer further discussion on these topics to Chapter 5.

3.2.1 Modifications and extensions

The formulation of f given thus far is accurate from a high-level standpoint — all the user really needs to be aware of is that there is some black box which is transforming their adjectival descriptions into procedural parameters. The exact details of how this is achieved are, however, more subtle.

Dimensionality and the inverse mapping

An immediate concern is the complexity of adjective space, since if adjective space were to have greater dimensionality than parameter space, then the use of adjectives will be unlikely to provide a simpler interface. Consider the 80 parameters employed by Weber and Penn [1995] — sifting through a list of 80 or more adjectives could be equally taxing, and for expert users probably quite frustrating. It is thus anticipated that adjective space will, in general, have lower dimensionality than parameter space — although this is is not an essential requirement.

With $\|\mathcal{A}\| < \|\mathcal{P}\|$, it is easily observed that f will not be *onto* — that is, there are some items of generated content with corresponding points $p \in \mathcal{P}$ such that $\forall a \in \mathcal{A}$ we have $f(a) \neq p$. More generally, suppose that there are no restrictions on the dimensionality of $\mathcal{A}$ and $\mathcal{P}$ — it is conceivable that there exist two items of generated content with corresponding points $p_1, p_2 \in \mathcal{P}$ with $p_1 \neq p_2$ such that both can be described by the same adjectival description $a \in \mathcal{A}$. It is then not possible for f to be onto, since we could have either $f(a) = p_1$ or $f(a) = p_2$ but not both since $p_1 \neq p_2$. It is also not possible to decide which of these two alternatives (if either) is a better choice, at least not without taking information regarding the specific procedural model into account. Closely related is the problem that some adjectival descriptions might not correspond to *any* generated content — in other words that there could exist some $a \in \mathcal{A}$ such that $\forall p \in \mathcal{P}$ we have $f(a) \neq p$. What content should be generated in the case where the user chooses such a description is unclear.

It is, however, reasonable to assume that *all* points $p \in \mathcal{P}$ give rise to generated content that can be described by some point $a \in \mathcal{A}$ — that is, each piece of generated content can be described in some way, even if such a description is simply to label the content as nondescript. Formally, we can say that $\forall p \in \mathcal{P}, \exists a \in \mathcal{A}$ such that $f^{-1}(p) = a$ and $\forall a_2 \in \mathcal{A}$ such that $a \neq a_2, f^{-1}(p) \neq a_2$. It thus turns out that f^{-1} is a much simpler function to work with conceptually, and this is our approach. Some extra work remains to give the user the illusion that they are working with the function f and this, as well as other additional benefits afforded by the inverse mapping, are discussed in more detail in Chapter 5.

Data amplification through semantic connectivity

A static set of adjectival descriptors in $\boldsymbol{A}$ will not necessarily be intuitive to all users — such a formulation is, in fact, quite subjective based as it is on a users' vocabulary and possibly their knowledge of specific topics. Even if the user is conversant with all the adjectives in $\boldsymbol{A}$, it is conceivable that they may wish to describe the content to be generated using their own descriptors.

To address these issues, the technique in this thesis makes use of the semantic connectivity data from WordNet [Fellbaum, 1998]. This allows alternative meanings for adjectival descriptors to be found, and also permits users to provide their own adjectives which can then be linked to known adjectival descriptors by using the semantic information present in WordNet. Further discussion on this aspect of the technique can be found in Chapter 5.

Varying importance of data

As discussed earlier, one key aspect that needs to be considered is the fact that different users will have different perceptions. Several avenues for addressing this problem were suggested, two of which involve the combination of data from multiple sources. When using data from multiple sources it is useful to be able to assign different levels of importance to them. In Chapter 5, a modification to a popular function approximation technique is presented that allows for the assignment of a weight to each data sample. This generalisation allows for varying importance to be placed on the input data, which in turn allows for the implementation of several of the strategies that have been proposed for dealing with differing perceptions.

3.2.2 Comparison and contrast to previous work

Due to the construction of adjective space in analogy to adverb space, the technique presented here is most similar to that of Rose et al. [1998]. Their focus is, however, specifically on stylised character motion as compared to the more generalised procedural modelling addressed by adjective space. As such, they focus on blending two emotional styles (that is, moving their adverb vector around an axis-aligned plane in adverb space), whereas adjective space allows for and encourages exploration of the space in a more general fashion. To be clear, adverb space is a specialised form of adjective space in which all the adjectives apply to verbs (and so are adverbs), and the technique presented in this thesis could be applied to the problem of stylised character motion if the underlying motion generation is expressed as a parametric procedural system.

Brand and Hertzmann [2000] make more explicit the ability to specify arbitrary style co-ordinates, although, since the styles are automatically learned, there is no natural language correspondence. Their use of HMMs is also very specific to the synthesis of motion which is time-dependent and relies on previous state information to realistically infer future state information.

Unuma et al. [1995] and Polichroniadis [2001] both differ substantially from the technique presented here, in that they seek to capture specific styles from example motions and then apply these to another piece of motion. In contrast, adjective space is used to map to procedural parameters which synthesise entirely new content — there is no attempt to transfer style from one example to another.

With reference to the natural language techniques that were discussed earlier, the technique presented draws inspiration from the work of Joshi et al. [2006] and Barnard and Forsyth [2001]. Similarly to Joshi et al., our technique makes use of WordNet [Fellbaum, 1998] to adapt more easily to the vocabualary level of the user. The use of an inverse mapping f^{-1} can also be viewed as related to Barnard and Forsyth's automatic annotation of images, as f^{-1} describes a function that produces a point in adjective space given a point in parameter space — that is, an automatic means for assigning an adjectival description to a piece of generated content.

3.3 Summary

In this chapter, we have posited an interface which is usable by novice or non-technical users for the procedural generation of complex digital content, and described the core mathematical framework through which the interface will operate. In the next chapter, we will explore the details of the mathematical framework with reference to related literature.

Chapter 4

Background to function approximation

As was introduced in Chapter 3, a mapping $f^{-1} : \mathcal{P} \rightarrow \mathcal{A}$ between parameter space and adjective space is sought. This is arguably the most critical part of the overall system, since without a good approximation even the most detailed procedural models will produce incorrect output in comparison to what the user desires.

Since the nature of f^{-1} is unknown, it is proposed that a number of points in $\mathcal{P}$ are pseudo-randomly chosen and the resulting content generated. For each piece of content, the user is given the opportunity to examine the content, after which they are asked to describe it using a set of pre-specified adjectives.

As a result of this learning phase, the system obtains a set $\mathcal{T} = \{\mathbf{t_1}, \mathbf{t_2}, \ldots, \mathbf{t_k}\} \subset \mathcal{P}$ of *training points* as well as the corresponding points in parameter space, $\mathcal{V} = \{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k}\} = \{f^{-1}(\mathbf{t_1}), f^{-1}(\mathbf{t_2}), \ldots, f^{-1}(\mathbf{t_k})\} \subset \mathcal{A}$.

Once $\mathcal{T}$ and $\mathcal{V}$ have been obtained, the system should be in a position to produce output $f^{-1}(x)$ for any $x \in \mathcal{P}$. There are several well-studied areas of research that can assist in approximating f^{-1}, namely *scattered data interpolation*, *scattered data approximation* and *online learning*. For each of these, there are a number of techniques that can be applied to solving the problem of approximating the function f^{-1}. This chapter focuses on considering the available techniques, and examining the situations for which they are best suited.

Before exploring related literature regarding function approximation techniques, one should first distinguish the subtle differences between the three classes of techniques introduced above:

- **Scattered data interpolation**. The important keyword here is *interpolation*, which suggests

that techniques in this class *must* interpolate the training data provided. In other words, once f^{-1} has been determined, then for all i in $1, 2, \ldots, k$ one must have $f^{-1}(\mathbf{t_i}) = \mathbf{v_i}$.

- **Scattered data approximation**. Collected data is usually prone to noise, and so often perfect interpolations of the data are either not possible or could be inaccurate. Scattered data approximation techniques aim to provide a good fit for data but at the sacrifice of possibly not perfectly interpolating the training data supplied. Usually, a sum squared error metric of the form

$$\sum_{i=1}^{k} \|f^{-1}(\mathbf{t_i}) - \mathbf{v_i}\|^2$$

  is minimised to give the function f^{-1} that best approximates the training data given.

- **Online learning**. Online learning techniques are characterised by the fact that they incrementally determine the function f^{-1} through successive sets of smaller training data (typically one input/output pair at a time). The advantage they offer over other techniques is that they can continually adjust the approximation using only the new data, whilst the other two classes of techniques typically require the entire set of all training data in order to form an approximation.

Since both $\mathcal{A}$ and $\mathcal{P}$ are subsets of some sets $\mathbf{R}^n$ and $\mathbf{R}^m$, in this chapter we refer to the function being approximated simply as f and address the general problem of function approximation between sets $\mathbf{R}^n$ and $\mathbf{R}^m$. It is also prudent to point out that the majority of function approximation techniques model functions of the form $f : \mathbb{R}^n \to \mathbb{R}$, slightly different from what adjective space requires which is the more general $f : \mathbb{R}^n \to \mathbb{R}^m$. There is evidence in the literature [Amidror, 2002] to suggest that the multi-valued approximation case is simply achieved by simultenaously solving m single-valued problems. When considering techniques in what follows, it is thus assumed that a function of the form $f : \mathbb{R}^n \to \mathbb{R}$ is being approximated from a training set of points $\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_k}$ and their corresponding scalar outputs $z_1, z_2, \ldots, z_k$.

4.1 Scattered data interpolation

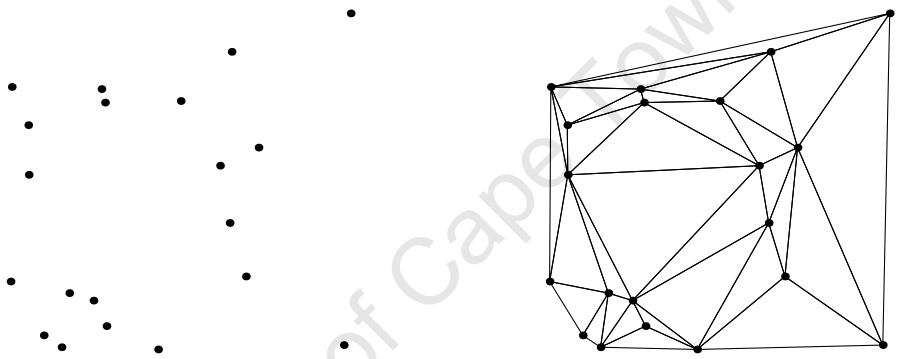
Amidror [2002] gives a good survey of interpolation techniques of the form $f : \mathbb{R}^2 \to \mathbb{R}$ and $f : \mathbb{R}^3 \to \mathbb{R}$. Some of these are explored in what follows, and for further details the reader should consult Amidror's work.

4.1.1 Triangulation based methods

In approximating a function about which nothing is known, *local methods* are often preferred — in which nearby points from the training data affect the output of the function. This is in contrast

to *global methods*, in which all of the training data has an impact on the output of the function. Using a local method thus typically preserves any localised changes in gradient that the function may exhibit, which may not be captured as well by a global method.

One of the simplest ways of guaranteeing a local method is to start by *triangulating* the training set. In two dimensions, this requires finding the maximal number of triangles whose vertices are in the training set, such that no triangle contains any point in the training set (consider, for example, Figure 31(a) which shows a set of points, and Figure 31(b) which shows a triangulation of the data). There are also several ways in which such maximal triangulations could be formed, but in practise *Delaunay* triangulations [Delaunay, 1934] are most often used, as they offer the additional guarantee that the circumcircle of each triangle will not contain any other points in the training set.



(a) The initial point-set (b) The triangulated point-set

Figure 31: *A point-set and its Delaunay triangulation*

With a triangulation in hand, consider now wishing to find the interpolant for some point $\mathbf{p}$. By finding the triangle in which $\mathbf{p}$ lies, one can use *barycentric coordinates* to describe $\mathbf{p}$ as a weighted sum of the three triangle vertices, and use these same weights to combine the function outputs at those vertices — giving the interpolated output of the function at $\mathbf{p}$.

Consider Figure 32 which shows a triangle defined by three points $\mathbf{p_1}$, $\mathbf{p_2}$ and $\mathbf{p_3}$, and the point $\mathbf{p}$ contained in the triangle. Constants a_1, a_2 and a_3 are sought such that $a_1\mathbf{p_1} + a_2\mathbf{p_2} + a_3\mathbf{p_3} = \mathbf{p}$, and $a_1+a_2+a_3 = 1$. If the coordinates of point $\mathbf{p_i}$ are given by (x_i, y_i) for $i = 1, 2, 3$ and $\mathbf{p} = (x, y)$, then the values of the a_i must satisfy

$$\begin{aligned} a_1x_1 + a_2x_2 + a_3x_3 &= x \\ a_1y_1 + a_2y_2 + a_3y_3 &= y \\ a_1 + a_2 + a_3 &= 1 \end{aligned}$$

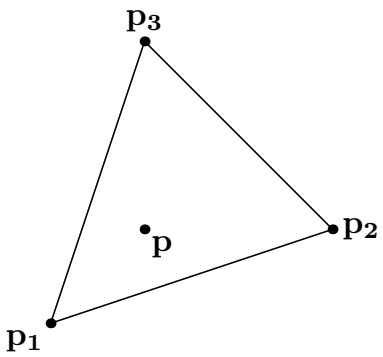


Figure 32: *Finding the barycentric co-ordinates of a point* $\mathbf{p}$ *within a triangle* $\mathbf{p_1p_2p_3}$.

and therefore have solution

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix}^{-1} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

The Cramer rule can then be used to eliminate the need for a full matrix inversion (see Amidror [2002] for details). With this solution one can now approximate $f(\mathbf{p}) = a_1 f(\mathbf{p_1}) + a_2 f(\mathbf{p_2}) + a_3 f(\mathbf{p_3})$.

But what of higher dimensions? Both adjective and parameter space could have more than two dimensions, and so one needs to address whether the Delaunay triangulation can be extended to higher dimensions, as well as whether a similarly neat barycentric coordinate solution exists for the a_i's. It is fairly easy to show that the second point holds, as in n dimensions one has $n+1$ equations to solve giving a final solution of

$$\begin{pmatrix} a_1 \\ \vdots \\ a_{n+1} \end{pmatrix} = \begin{pmatrix} p_{1,1} & \cdots & p_{(n+1),1} \\ \vdots & \ddots & \vdots \\ p_{1,n} & \cdots & p_{(n+1)_n} \\ 1 & \cdots & 1 \end{pmatrix}^{-1} \begin{pmatrix} p_1 \\ \vdots \\ p_n \\ 1 \end{pmatrix}$$

where $p_{i,j}$ indicates the j^{th} component of $\mathbf{p_i}$ and p_j indicates the j^{th} component of $\mathbf{p}$. So a solution exists, provided that the matrix of equation coefficients is invertible — which can only occur if all the points $\mathbf{p}$ lie on a $(n-1)$-dimensional plane, an impossibility if these points are the vertices in a Delaunay triangulation.

The problem of higher-dimensional Delaunay triangulations is somewhat trickier. First, one must address what constitutes a triangle in higher dimensions. In three dimensions, this is a tetrahedron and algorithms for finding a three-dimensional triangulation, or *tetrahedrisation*, are fairly common and well used. In general, the analogue of a two-dimensional triangle in an arbitrary number of dimensions d is referred to as a *d-simplex*, and for a given set of points $P \subset \mathbb{R}^d$, a d-simplex is defined as the convex combination of $d+1$ affinely independent points in P. It has been shown in the literature [Scopigno, 1998] that the upper bound on the number of simplices produced by a Delaunay triangulation is $O(n^{\lceil \frac{d}{2} \rceil})$, and consequently this also serves as a best-case lower bound for the algorithmic complexity of computing these simplices. Table 1 gives the computed upper bound on the number of simplices for a range of dataset sizes and dimensions: as can be seen, the explosion in both space and time complexity for fairly modest data-set size increases is enormous.

Therefore, without significant advances in computational power and massive memory storage, it seems that using such a technique for higher dimensions may be infeasible.

One of the failings irrespective of dimension for the method described above is that, due to the

	3 dimensions	5 dimensions	8 dimension	10 dimensions
20 points	400	8000	160,000	3,200,000
50 points	2,500	125,000	6,250,000	312,250,000
100 points	10,000	1,000,000	100,000,000	10,000,000,000
500 points	250,000	12,500,000	625,000,000	31,225,000,000

Table 1: *Upper bounds on the number of simplices produced by Delaunay triangulation for the given numbers of points and dimensions.*

piecewise linear nature of the interpolation as one crosses simplex boundaries, C^1 continuity is lost. In a number of applications having such continuity is important and, as a result, a number of extensions have been developed that still make use of the underlying triangulation but perform the interpolation step slightly differently. The most popular and widely used is the Clough-Tocher method [Clough and Tocher, 1965], which utilises bivariate cubic polynomials within each triangle to produce a C^1-continuous interpolant. As this method also relies on a triangulation, which has been shown to be unsuitable for higher dimensions, further details are omitted (see Amidror [2002]).

4.1.2 Natural neighbour interpolation

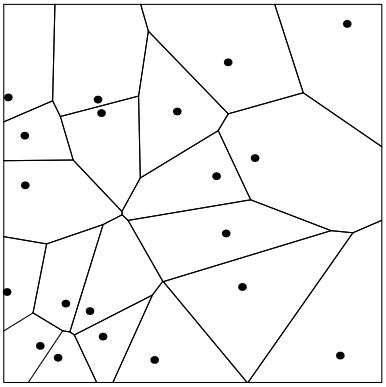
Strongly related to the triangulation based methods is the concept of *natural neighbour interpolation*, introduced by Sibson [1981]. Where triangulation based schemes make use of the triangulation of a set of data, natural neighbour interpolation deals with the dual of triangulation: known as a *Voronoi diagram* [Voronoi, 1907] or *Dirichlet tessellation* [Dirichlet, 1850], and also referred to as a *Voronoi tessellation*. Given the points $\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n}$, the Voronoi region or tile T_i is the set of all points that are closer to $\mathbf{p_i}$ than to any other $\mathbf{p_j}$

$$T_i = \{\mathbf{x} \in \mathbb{R}^n \mid \forall j \neq i : \|\mathbf{x} - \mathbf{p_j}\| > \|\mathbf{x} - \mathbf{p_i}\|\}$$

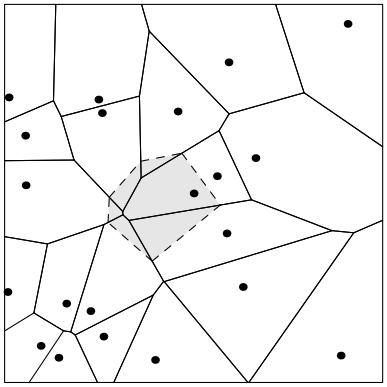
Each tile is an open, convex polyhedron, and the union of all the tiles covers the entire domain $\mathbb{R}^n$ except for the boundaries of the tiles (see Figure 33(a) for an example of a Voronoi tessellation and the associated tiles). Tiles that share a common boundary are referred to as *neighbours*, as are the data points associated with the tiles[1]. A quantitative relationship between these neighbours is then established by considering the subtiles T_{ij}, where T_{ij} is the portion of T_i that has $\mathbf{p_j}$ as its second-nearest neighbour (that is, the points $\mathbf{x}$ such that if $\mathbf{p_i}$ were removed and the Voronoi diagram recomputed then $\mathbf{x}$ would belong to T_j). By writing $\kappa(i)$ for the Lebesgue measure[2] [Lebesgue, 1902; Stein and Shakarchi, 2005] of the tile T_i and $\kappa_j(i)$ for that of T_{ij}, one can consider the normalised form $\lambda_j(i) = \frac{\kappa_j(i)}{\kappa(i)}$ as a measure of how "strong" $\mathbf{p_j}$ is as a neighbour of $\mathbf{p_i}$. Figure 33(b) illustrates how a tile in a Voronoi tessellation may be divided into subtiles.

Having established a means for relating data points to each other quantitatively, Sibson goes on to

[1] By joining data points that are neighbours with a line, the dual Delaunay triangulation of the data is obtained.
[2] The length, area or volume of a set of points in Euclidean space.



(a) An example of a Voronoi diagram



(b) The shaded area shows the tile which a new point would generate if added to the Voronoi diagram in (a). The intersection of this tile with each of the existing tiles gives the subtiles associated with the point.

Figure 33: *(a) A Voronoi diagram, and (b) the subtiles generated by considering a point not in the diagram.*

describe how a relationship between an arbitrary point $\mathbf{p}$ and the data points $\mathbf{p_i}$ can be established. This is simply achieved by adding $\mathbf{p}$ to the tessellation, and extending our Lebesgue measure notation to include this new point by writing $\kappa(\mathbf{p})$ to indicate the Lebesgue measure of the tile associated with $\mathbf{p}$, $\kappa_j(\mathbf{p})$ to indicate the area of the subtile implied by $\mathbf{p_j}$, and $\lambda_j(\mathbf{p}) = \frac{\kappa_j(\mathbf{p})}{\kappa(\mathbf{p})}$. The normalised values $\lambda_j(\mathbf{p})$ are referred to as *local coordinates*, and satisfy two particularly important properties:

1. $\lambda_j(\mathbf{p})$ is continuous for all j.

2. The $\lambda_j(\mathbf{p})$ satisfy the *local coordinates property*, which states that $\sum \lambda_j(\mathbf{p})\mathbf{p_j} = \mathbf{p}$.

With these properties in hand, the *natural neighbour C^0 interpolant* can be constructed as

$$f^{(0)}(\mathbf{p}) = \sum \lambda_j(\mathbf{p}) z_j$$

Unfortunately, $f^{(0)}$ suffers from discontinuities at the $\mathbf{p_i}$. By estimating the gradient so as to enforce neighbouring points to lie on a spherical quadric, some rigorous calculation yields the *natural neighbour $C^{(1)}$ interpolant*

$$f^{(1)}(\mathbf{p}) = \frac{\frac{\sum \lambda_j(\mathbf{p}) d_j(\mathbf{p})}{\sum [\lambda_i(\mathbf{p})/d_i(\mathbf{p})]} f^{(0)}(\mathbf{p}) + \sum [\lambda_j(\mathbf{p}) d_j^2(\mathbf{p})] \zeta(\mathbf{p})}{\frac{\sum \lambda_j(\mathbf{p}) d_j(\mathbf{p})}{\sum [\lambda_j(\mathbf{p})/d_j(\mathbf{p})]} + \sum \lambda_j(\mathbf{p}) d_j^2(\mathbf{p})}$$

where

$$
\begin{aligned}
d_j(\mathbf{p}) &= \sqrt{(\mathbf{p}-\mathbf{p_j})^\top(\mathbf{p}-\mathbf{p_j})} \\
\zeta(\mathbf{p}) &= \alpha + \beta^\top \mathbf{p} + \gamma \mathbf{p}^\top \mathbf{p} - \gamma \frac{\sum \lambda_j(\mathbf{p}) d_j(\mathbf{p})}{\sum [\lambda_i(\mathbf{p})/d_i(\mathbf{p})]}
\end{aligned}
$$

The coefficients of $\zeta(\mathbf{p})$ are determined from the assumption that $\mathbf{p}$ and its neighbours $\mathbf{p_j}$ lie on a spherical quadric, giving $z_j = \alpha + \beta^\top \mathbf{p_j} + \gamma \mathbf{p_j}^\top \mathbf{p_j}$.

Following on from the work of Sibson, Christ et al. [1982] proposed the use of the so-called *boundary-to-distance* weight as an alternative for measuring the contribution of each input point to the interpolation. This yields the *Laplace interpolant*, whose local coordinates are given by

$$
\lambda_j(\mathbf{p}) = \frac{\alpha_j(\mathbf{p})}{\sum_k \alpha_k(\mathbf{p})},\ \alpha_j(\mathbf{p}) = \frac{s_j(\mathbf{p})}{h_j(\mathbf{p})}
$$

where $s_j(\mathbf{p})$ is the Lebesgue measure of the Voronoi edge associated with the j^{th} natural neighbour of $\mathbf{p}$, and $h_j(\mathbf{p})$ is the distance between $\mathbf{p}$ and the j^{th} natural neighbour of $\mathbf{p}$ (see Figure 34 for an example). Similarly to Sibson's interpolant, the Laplace interpolant can be extended to afford higher levels of continuity.

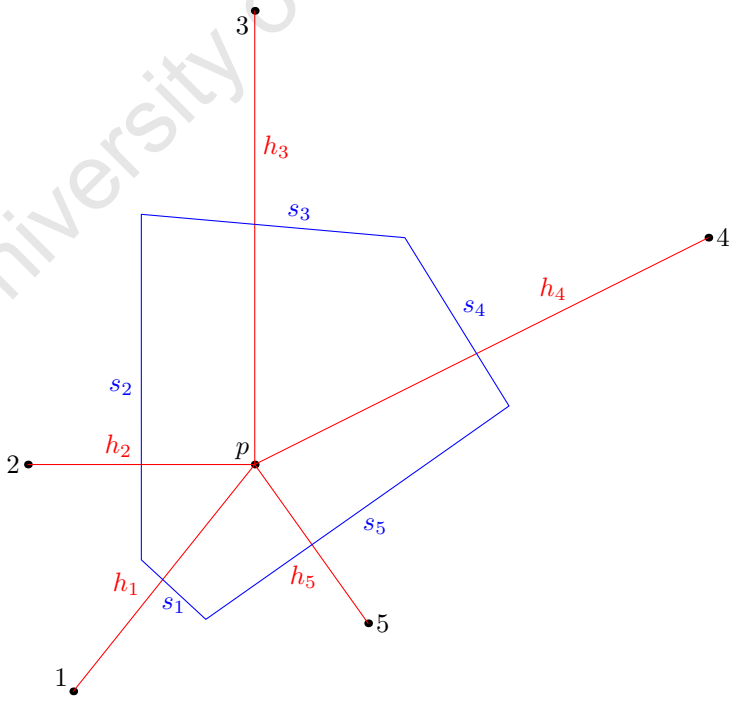


Figure 34: *An illustration of the quantities used in the computation of the Laplace interpolant.*

However, despite the C^1 continuity benefits afforded by the Sibson and Laplace interpolants, they become unwieldy to compute in higher dimensions due to their reliance on the Voronoi tessellation.

The vertices defining the tiles T_i are points that are equidistant from $d+1$ of the data points, and therefore correspond to the circumcentres of the simplices defined by the data points in the Delaunay triangulation of the data. The number of vertices is therefore equal to the number of simplices, and so the complexity of natural neighbour interpolants grows according to the values in Table 1. Furthermore, computing the Lebesgue measure for sets in higher dimensions is non-trivial and so, unfortunately, this technique has little use for data of dimension greater than 3.

4.1.3 Inverse-distance based methods

Shepard interpolants are some of the most commonly used methods for interpolation, and are based on using the inverse distance to the points in the training set as weights for the interpolation [Shepard, 1968]. The core idea is that the output of the function at any point should be influenced more by training data nearer to that point, and less so by more distant points. There are a number of different ways for choosing the weights, which will now be explored.

Although Shepard's proposal was for the two-dimensional case, it is very easily generalised to an arbitrary number of dimensions. With a training set of n points $\mathbf{p_i}$ whose outputs are z_i for $i = 1, 2, \ldots, n$, Shepard's interpolant can be expressed as

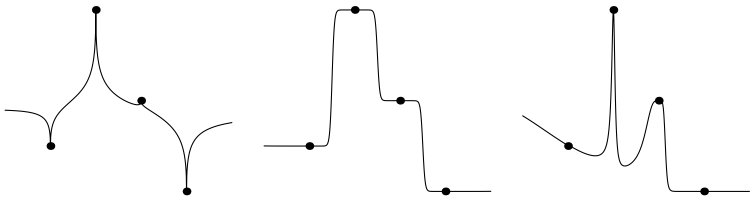
$$f(\mathbf{p}) \quad = \sum_{i=1}^{n} w_i(\mathbf{p}) z_i = \sum_{i=1}^{n} \frac{h_i(\mathbf{p})}{\sum_{i=1}^{n} h_i(\mathbf{p})} z_i \tag{1}$$

where

$$h_i(\mathbf{p}) = \frac{1}{d_i(\mathbf{p})^k}$$

and $d_i(\mathbf{p})$ is the Euclidean distance between $\mathbf{p}$ and $\mathbf{p_i}$ for some pre-chosen exponent $k \geq 1$. Additionally it should be noted that $h_i(\mathbf{p_i})$ is infinitely large, but it can be shown that as $\mathbf{p}$ approaches $\mathbf{p_i}$, $f(\mathbf{p})$ approaches z_i and so explicitly setting $f(\mathbf{p_i}) = z_i$ gives a continuous function. Furthermore, with the weight functions as specified in Equation 1, one obtains the additional useful properties that $0 \leq w_i(\mathbf{p}) \leq 1$ and $\sum_{i=1}^{n} w_i(\mathbf{p}) = 1$.

One problem with Shepard interpolants is that smaller values of k can give too much influence to distant points (see, for example, Figure 35(a), which also exhibits slope discontinuities that may be undesirable in some circumstances). Increasing the k exponent in the h_i functions has a smoothing effect, as can be seen in Figure 35(b), ultimately causing the function to tend towards a step function as $k \to \infty$. Additionally, one could set a different k value for each h_i: an example of this is shown in Figure 35(c).



(a) $k = 0.5$ (b) $k = 10$ (c) $k = 1, 2, 5, 10$

Figure 35: *An example of how varying exponents in Shepard interpolants affects the overall smoothing of the approximation.*

To make use of the locality benefits afforded by higher k exponents but eradicate the flattening around each of the $\mathbf{p_i}$ points, one can replace the z_i values in Equation 1 with a local approximation $L_i z(\mathbf{p})$ to f around the point $\mathbf{p_i}$. If gradient data is either available or can be estimated, then a good choice for $L_i z$ is

$$L_i z(\mathbf{p}) = z_i + \delta_{\mathbf{i}}(\mathbf{p} - \mathbf{p_i})$$

where $\delta_{\mathbf{i}}$ is a vector of the partial derivatives at $\mathbf{p_i}$. The effect that this has on the interpolant can be seen in Figure 36. Additionally, because Shepard interpolants are, by their nature, global, they can be too slow for large datasets, and some ways of adjusting them to be more local have been suggested. One possibility is to modify the h_i functions such that

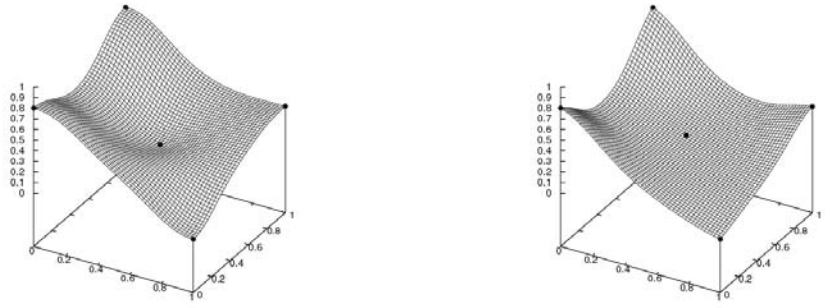
$$h_i(\mathbf{p}) = \left\{ \frac{[R - d_i(\mathbf{p})]_+}{R d_i(\mathbf{p})} \right\}^k$$

for a pre-defined radius of influence R around each point in the training set, where $[R - d_i(\mathbf{p})]_+$ is the piecewise function defined as

$$[R - d_i(\mathbf{p})]_+ = \begin{cases} R - d_i(\mathbf{p}) & \text{if } d_i(\mathbf{p}) < R \\ 0 & \text{if } d_i(\mathbf{p}) \geq R \end{cases}$$

Reformulating the h_i functions in this way restricts the influence of each data point to a constrained area.

In summary, most interpolation methods are poor choices for applying to adjective space due to their reliance on a Delaunay triangulation or Voronoi tessellation. Shepard interpolants are free from this constraint, but additionally require gradient information and fine-tuning of the exponent valus.



(a) No gradient information (b) With estimated gradient information

Figure 36: *The effect of using estimated gradient information to obtain a more local approximation around the supplied data points.*

4.2 Scattered data approximation

Scattered data approximation techniques relax the requirements of interpolation methods, by allowing for a function that does not necessarily interpolate the given training points (although it is possible that interpolation of all the training points may occur). Instead, scattered data approximation methods focus on achieving a "better" fit to the data, typically through the use of higher-order functions than those employed by interpolation techniques.

4.2.1 Least squares, weighted least squares, and moving least squares

Three scattered data approximation techniques that are extremely similar in nature are *least squares*, *weighted least squares* and *moving least squares*. They all build on the same basic formulation, with the following defining characteristics:

- **Least squares** gives a global approximation
- **Weighted least squares** gives a local approximation computed at discrete points
- **Moving least squares** gives a local approximation computed continuously over the domain

Least squares

Least squares seeks to approximate f by minimising the mean squared error between f and the outputs z_i at each point $\mathbf{p_i}$, given by the error function

$$C(f) = \sum_{i=0}^{n} \|f(\mathbf{p_i}) - z_i\|^2 \tag{2}$$

which needs to be minimised. In a typical least squares system, f is chosen from $\prod_m^d$, the space of polynomials in d dimensions of total degree m, and so f can be written as

$$f(\mathbf{p}) = \mathbf{b}(\mathbf{p}) \cdot \mathbf{c}$$

where $\mathbf{b}(\mathbf{x})$ is the polynomial basis vector $[b_1(\mathbf{x}), \ldots, b_k(\mathbf{x})]^\top$ and $\mathbf{c}$ is the vector of unknown coefficients $[c_1, \ldots, c_k]^\top$ that will be solved for. For example, if $\mathbf{x} = (x, y)$ (i.e. $d = 2$) and $m = 2$, then the polynomial basis vector $\mathbf{b}$ would be $[1, x, y, x^2, xy, y^2]^\top$. For functions f of this form taken from $\prod_m^d$, the number of elements in $\mathbf{b}$ is $k = \frac{(d+m)!}{m!d!}$ [Levin, 1998].

The error in Equation 2 is minimised by taking the partial derivatives of $C(f)$ with respect to the unknown constants $c_1, \ldots, c_k$ and setting each of these to be equal to zero, giving k simulatenous equations of the form

$$\begin{aligned} \frac{\partial C}{\partial c_1} = 0 \quad &\Rightarrow \quad \sum_{i=1}^{n} 2b_1(\mathbf{p_i}) \left[\mathbf{b}(\mathbf{p_i})^\top \mathbf{c} - z_i\right] = 0 \\ &\vdots \\ \frac{\partial C}{\partial c_k} = 0 \quad &\Rightarrow \quad \sum_{i=1}^{n} 2b_k(\mathbf{p_i}) \left[\mathbf{b}(\mathbf{p_i})^\top \mathbf{c} - z_i\right] = 0 \end{aligned}$$

which can be grouped into the single matrix-vector equation

$$2\sum_{i=1}^{n} \mathbf{b}(\mathbf{p_i}) \left[\mathbf{b}(\mathbf{p_i})^\top \mathbf{c} - z_i\right] = \mathbf{0} \Rightarrow \sum_{i=1}^{n} \left[\mathbf{b}(\mathbf{p_i})\mathbf{b}(\mathbf{p_i})^\top \mathbf{c} - \mathbf{b}(\mathbf{p_i})z_i\right] = \mathbf{0}$$

Via some rearrangement, the solution is found to be

$$\mathbf{c} = \left[\sum_{i=1}^{n} \mathbf{b}(\mathbf{p_i})\mathbf{b}(\mathbf{p_i})^\top\right]^{-1} \sum_{i=1}^{n} \mathbf{b}(\mathbf{p_i})z_i$$

or alternatively

$$\mathbf{c} = \left(\mathbf{B}^\top \mathbf{B}\right)^{-1} \mathbf{B}^\top \mathbf{z} \tag{3}$$

where $\mathbf{B}$ and $\mathbf{z}$ are given by

$$\mathbf{B} = \begin{pmatrix} b_1(\mathbf{p_1}) & \dots & b_k(\mathbf{p_1}) \\ \vdots & \ddots & \vdots \\ b_1(\mathbf{p_n}) & \dots & b_k(\mathbf{p_n}) \end{pmatrix} \text{ and } \mathbf{z} = \begin{pmatrix} z_1 \\ \vdots \\ z_k \end{pmatrix}$$

Clearly the feasibility of a solution to Equation 3 is dependent on the matrix $\mathbf{B}^\top \mathbf{B}$ being nonsingular and thus invertible, but this is easy to check for. In most cases, provided that $n \geq k$ the matrix can be inverted and a solution for the constants found.

Weighted least squares

In order to localise the least squares fit described previously, the error function in Equation 2 is weighted with a distance-based function $\theta(\|\bar{\mathbf{p}} - \mathbf{p_i}\|)$ for some fixed point $\bar{\mathbf{p}}$ to give

$$C_{\bar{\mathbf{p}}}(f) = \sum_{i=1}^{n} \theta(\|\bar{\mathbf{p}} - \mathbf{p_i}\|) \|f(\mathbf{p_i}) - z_i\|^2 \tag{4}$$

To achieve a local fit to the data, the weighting function $\theta(d)$ is typically chosen such that $\theta(0) \approx 1$ and $\theta(d) \approx 0$ for all $d > h$, where h is the *effective radius* of the function. Varying h thus affects how local the fit is by varying the radius supported by the function: this is discussed in more detail in Section 4.2.2. Common choices for θ include the Gaussian

$$\theta(d) = e^{\frac{d^2}{h^2}}$$

and the Wendland function [Wendland, 2005]

$$\begin{aligned} \theta(d) &= (1 - \tfrac{d}{h})^4 (4\tfrac{d}{h} + 1) && \text{if } d \leq h \\ \theta(d) &= 0 && \text{if } d > h \end{aligned}$$

Since the only change between Equations 2 and 4 is the addition of the extra weight term, the derivation of the unknown constants remains the same. The only major difference to bear in mind is that these constants are dependent on the fixed point $\bar{\mathbf{p}}$, and so we write them as $\mathbf{c}(\bar{\mathbf{p}})$ and denote

the local function f defined by these constants as $f_{\bar{\mathbf{p}}}$.

How now does one apply this local weighted least squares solution to obtain a global approximation over the entire domain? The solution is to combine a number of these local functions, such that every point in the domain is supported by at least one of the local functions. For points that are supported by more than one local function, a *partition of unity* [Shepard, 1968] can be used to determine an appropriate weighting for each local function. Suppose that we have l such local functions defined by the fixed points $\bar{\mathbf{p}}_\mathbf{j}, j \in [1, \ldots, l]$, then the weight for each function is defined as

$$\psi_j(\mathbf{p}) = \frac{\theta(\|\bar{\mathbf{p}}_\mathbf{j} - \mathbf{p}\|)}{\sum_{k=1}^{l} \theta(\|\bar{\mathbf{p}}_\mathbf{k} - \mathbf{p}\|)}$$

and the global approximating function is given by

$$f(\mathbf{p}) = \sum_{j=1}^{l} \psi_j(\mathbf{p}) \mathbf{b}(\mathbf{p})^\top \mathbf{c}(\bar{\mathbf{p}}_\mathbf{j})$$

Moving least squares

Varying density in the training data can make the weighted least squares approach above slightly unwieldy to use, as good choices for the number and positioning of the fixed weighting points is not necessarily intuitive. Moving least squares [Lancaster and Salkauskas, 1981; Levin, 1998] solves this by using a single localised weighted least squares approach, where the fixed point is the actual point at which we wish to calculate the output of the function, giving

$$f(\mathbf{p}) = f_\mathbf{p}(\mathbf{p}) = \mathbf{b}(\mathbf{p})^\top \mathbf{c}(\mathbf{p})$$

In other words, for every point $\mathbf{p}$ at which f needs to be evaluated, $\mathbf{c}(\mathbf{p})$ must be calculated using the weighted least squares solution. Although this can be an expensive process, finding a suitable choice for h in the weighting function θ will ensure that the rank of $\mathbf{B}$ in the linear system is substantially decreased, making for a significant speedup. Furthermore, due to the local nature and density agnosticism of moving least squares, it will in almost all cases produce the best function approximation results and thus should be preferred over the standard weighted or unweighted variants where possible.

In contrast to triangulation and natural neighbour techniques, the least squares methods are much more extensible to higher dimensions. The only place where the number of dimensions plays a role is in the number of coefficients that are solved for, which is given by $k = \frac{(d+m)!}{m!d!}$ — this, in turn, places a lower bound on the rank of $\mathbf{B}$ to ensure that a unique solution can be found, which enforces the same lower bound on the number of data points provided to the algorithm as input. Some examples

of the value of k for various combinations of d and m are shown in Table 2 — which is useful in cases where data is elicited from users and therefore where one must be mindful of the quantity of data that would be required. Typically $m \leq 2$, but even then some of the higher dimensions might require too much data to reasonably expect a user to provide.

	d=2	d=3	d=5	d=10	d=15	d=20	d=30
m=1	3	4	5	10	15	20	30
m=2	6	10	21	66	136	231	496
m=3	10	20	56	286	816	1771	5456

Table 2: *The number of coefficients k that need to be solved for in a least squares system that takes its polynomial basis vector from $\prod_m^d$, for varying values of d and m.*

4.2.2 Radial basis function networks (RBFNs)

Radial basis function networks [Broomhead and Lowe, 1988] are one of the more popular methods for scattered data approximation. Although they are sometimes classed as interpolation schemes, in practise RBFNs are usually unable to interpolate their training data but rather give the best fit to the data in a least-squares sense, as will be demonstrated.

A traditional radial basis function network (RBFN) is a special case of the more general *neural network*[3], with the following defining characteristics:

- An RBFN has exactly one hidden layer. Typically the number of units in the hidden layer is equal to the number of points in the training set — where each unit models the contribution of its corresponding point in the training set — but this need not be the case.
- Each unit in the hidden layer is modelled by a *radial basis function* (RBF), and all components of the input vector $\mathbf{x}$ are fed forward into every unit in the hidden layer.
- The outputs from the hidden layer are linearly combined with weights to form the function's output.

A graphical illustration of an RBFN is shown in Figure 37.

The choice of the radial basis functions used is typically the hardest part in finding a good approximation, and is largely dependent on the density of the underlying data as well as the local or global nature of the resulting approximant. Common choices for radial basis functions are the Gaussian and Wendland functions defined earlier, and the multiquadric

$$\theta(d) = \frac{\sqrt{h^2 + d^2}}{h}$$

[3]Neural networks are discussed in greater depth in Section 4.3.1

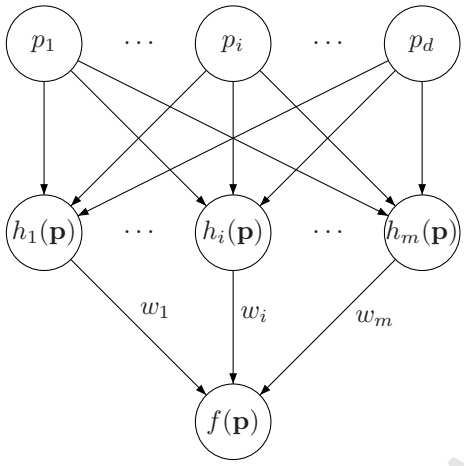


Figure 37: *A traditional RBFN accepts input from an d-dimensional vector,* $\mathbf{p}$*, which is fed forward to the m hidden units* h_i *(*$i \in 1 \ldots m$*). The outputs from the hidden units are then each weighted by* w_i *and summed to give the result,* $f(\mathbf{p})$*.*

where d denotes the distance $\|\mathbf{p}-\mathbf{c}\|$ of an arbitrary point $\mathbf{p}$ from the center of the function $\mathbf{c}$, and h is a parameter used to influence the radius supported by the function. The Gaussian and Wendland functions are good examples of locally supporting functions, as at $d = 0$ they have value 1 but as d increases their value approaches 0 (and for the Wendland function, $\theta(d) = 0$ for $d \geq h$); whilst the multiquadric gives more significant output for higher values of d. By manipulating the parameter h, one is also able to adjust exactly how local the functions are — for a sufficiently large value of h a global approximation can be achieved. It is also possible to choose different radial basis functions and parameters h for each of the hidden units of the RBFN — this has no effect on the mathematics that follow, which explore RBFNs in more detail. In general, the i^{th} hidden unit centred at $\mathbf{c_i}$ is referred to as h_i, where $h_i(\mathbf{p}) = \theta_i(\|\mathbf{p} - \mathbf{c_i}\|)$ for some RBF θ_i.

From Figure 37, the mathematical forumlation of an RBFN function f can be written as

$$f(\mathbf{p}) = \sum_{i=1}^{m} w_i h_i(\mathbf{p})$$

and, since the h_i functions are pre-specified, the only unknowns are the weights w_i which need to be solved for. For f to approximate the training data as closely as possible, one has n equations of the form

$$z_j \approx \sum_{i=1}^{m} w_i h_i(\mathbf{p_j})$$

and the total error between f and the training data can be expressed in a least squares sense as

$$C = \sum_{j=1}^{n} \|f(\mathbf{p_j}) - z_j\|^2 \tag{5}$$

Equation 5 should look quite familiar — identical to Equation 2, in fact. And the solution is identical: by taking partial derivatives and equating these to 0, some rearrangement yields the solution for the unknown constants, $\mathbf{w}$, as

$$\mathbf{w} = \left(\mathbf{H}^\top \mathbf{H}\right)^{-1} \mathbf{H}^\top \mathbf{z}$$

where $\mathbf{H}$, termed the *design matrix*, is given by

$$\mathbf{H} = \begin{pmatrix} h_1(\mathbf{p_1}) & \cdots & h_m(\mathbf{p_1}) \\ \vdots & \ddots & \vdots \\ h_1(\mathbf{p_n}) & \cdots & h_m(\mathbf{p_n}) \end{pmatrix}$$

RBFNs are considered a generally good option for effective function approximation, as different choices for the radial basis functions and their radii of support allow the network to adapt well to many classes of underlying functions, as well as provide both local and global approximations. Furthermore, RBFNs can be incrementally altered (that is, additional training points can be added or existing training points removed without having to re-solve the entire linear algebra system) — see Orr [1996] for details[4].

As they have been presented thus far, RBFNs can be prone to numerical instability. One means for solving this is by a imposing regularisation penalty on the weights, by modifying the cost function to

$$C = \sum_{j=1}^{n} \|f(\mathbf{p_j}) - z_j\|^2 + \sum_{i=1}^{m} \lambda_i w_i^2 \tag{6}$$

which gives the solution

$$\mathbf{w} = \left(\mathbf{H}^\top \mathbf{H} + \mathbf{\Lambda}\right)^{-1} \mathbf{H}^\top \mathbf{z}$$

where $\mathbf{\Lambda}$ is zero everywhere except for the diagonal, which contains the regularisation parameters

[4] Although this distinction should technically categorise RBFNs as a form of online learning, it is in fact a feature which is not often utilised, since most applications only need to solve for the weights of the network once. As such, the commonly accepted categorisation for RBFNs is in the class of scattered data approximation techniques.

λ_j. Often a single regularisation parameter λ is used, in which case $\mathbf{\Lambda} = \lambda\mathbf{I}$; we provide the more general form above for completeness. In the event that the function being modelled is guaranteed or desired to be always positive, then it is also possible to employ a *non-negative least squares* solution [Lawson and Hanson, 1974], which iteratively minimises the cost in Equation 5 by choosing only non-negative values for the weights w_i.

Model selection criteria and $\mathbf{\Lambda}$ optimisation

Due to the extra weight penalty added by the regularisation parameters in Equation 6, they do more than just stabilise the numerical solution. Smaller regularisation parameter values allow for a tighter fit to the data at the expense of potentially large changes in surface curvature, whilst larger values impose a smoother curve at the expense of a poorer data fit. Choosing suitable regularisation parameters is therefore quite important, and is achieved by minimising what are known as *model selection criteria* (MSC) [Orr, 1996, 1999]. The goal of model selection criteria are to measure error in a model by providing an estimate of how well the model will perform when given later input (which may or may not be different to inputs in the training set) — minimising a model selection criterion means effectively minimising the estimate for error on future data sets, and thus achieving the best balance between smoothness and fitting of the training data.

For problems in which there is an abundance of data, the simplest way of optimising the model is to partition the data into two sets — one for training, and one for testing. Large quantities of data are not always available, however, and there is the possibility that any particular partitioning could bias the solution. An improvement on this is to try several different partitionings of the data and then take the average of the results. Taken to the extreme, one could consider a training set of $n-1$ samples and a test corpus of 1 sample, observing the squared error that the model exhibits for the test sample. By considering the n different ways of creating such a partition and averaging these squared errors, one calculates what is known as the *leave-one out (LOO)* estimator.

There are two major advantages to an estimator such as LOO: first and foremost, no data has to be sacrificed and put aside as a specific testing set — *all* the data is used to both train and test the model. Secondly, there is an elegant calculation for LOO that avoids the need of training and testing n separate models. Consider a model that is trained using all n samples. The LOO sum-squared error can then be written as

$$\sigma^2_{\mathrm{LOO}} = \frac{\mathbf{z}^\top \mathbf{P}(\mathrm{diag}(\mathbf{P}))^{-2}\mathbf{P}\mathbf{z}}{n}$$

where

$$\mathbf{P} = \mathbf{I} - \mathbf{H}(\mathbf{H}^\top\mathbf{H} + \mathbf{\Lambda})^{-1}\mathbf{H}^\top$$

is known as the *projection matrix* (see Orr [1996]). A number of simple operations (such as adding or removing a basis function, or adding or removing a point of training data) can be easily modelled through changes to the projection matrix.

The matrix diag($\mathbf{P}$) makes LOO slightly unwieldy, and so in practise the related criteria known as the *generalised cross-validation* (GCV) [Golub et al., 1979] and *Bayesian information criterion* (BIC) [Schwarz, 1978] are used instead, which can be calculated as

$$\begin{aligned} \sigma^2_{\text{GCV}} &= \frac{n\mathbf{z}^\top\mathbf{P}^2\mathbf{z}}{(n-\gamma)^2} \\ \sigma^2_{\text{BIC}} &= \frac{n+(\ln(n)-1)\gamma}{n-\gamma}\frac{\mathbf{z}^\top\mathbf{P}^2\mathbf{z}}{n} \end{aligned}$$

where $\gamma = n - \text{trace}(\mathbf{P})$, and the common term $\mathbf{z}^\top\mathbf{P}^2\mathbf{z}$ gives the sum-squared error of the model over the training set. Armed with a model selection criterion, the next step is to find suitable values for the regularisation parameters. In the event that a single parameter λ is used, setting the derivative of the MSC to 0 allows one to rearrange the resulting equation and obtain a re-estimation formula for λ [Orr, 1996]. For GCV and BIC, these re-restimation formulae are

$$\begin{aligned} \text{GCV: } \hat{\lambda} &= \frac{\mathbf{z}^\top\mathbf{P}^2\mathbf{z}\ \text{trace}\left(\mathbf{A}^{-1}-\lambda\mathbf{A}^{-2}\right)}{\mathbf{w}^\top\mathbf{A}^{-1}\mathbf{w}}\frac{1}{n-\gamma} \\ \text{BIC: } \hat{\lambda} &= \frac{\mathbf{z}^\top\mathbf{P}^2\mathbf{z}\ \text{trace}\left(\mathbf{A}^{-1}-\lambda\mathbf{A}^{-2}\right)}{\mathbf{w}^\top\mathbf{A}^{-1}\mathbf{w}}\frac{n\log(n)}{2(n-\gamma)(n+(\log(n)-1)\gamma)} \end{aligned}$$

By repeatedly re-estimating λ until convergence, the criterion is minimised in much the same way as Newton's method uses re-estimation to converge to the solution of an equation. The only complicating factor with these formulations is their reliance on the inverse of the matrix $\mathbf{A}$, an expensive operation that is prone to numerical instability. Orr's later work [1999] improves on this re-estimation process by rewriting the formulae using the eigenvectors and eigenvalues of $\mathbf{H}\mathbf{H}^\top$ — resulting in a once-off *singular value decomposition* [Golub and Van Loan, 1996a] and a linear cost in n for re-estimation.

The trouble with a single regularisation parameter is that it globally controls the smoothness of a function — whilst this may work well in general, some functions may exhibit locally differing degrees of smoothness, and in these cases it is desirable to control the regularisation on a local level. It was mentioned earlier that RBFNs can be incrementally altered through changes to the projection matrix: this in fact provides the basis for local regularisation. In particular, Orr shows that the addition of a new basis function to the network causes $\mathbf{P}$ to be updated to

$$\mathbf{P_j} = \mathbf{P} - \frac{\mathbf{P h_j h_j}^\top \mathbf{P}}{\lambda_j + \mathbf{h_j}^\top \mathbf{P h_j}}$$

where h_j is the column that would be appended to the design matrix as the result of adding this basis function, that is $h_j = [h_j(\mathbf{p_1}) \cdots h_j(\mathbf{p_n})]^\top$. Using this, Orr derives an analytical method for the optimal value of λ_j which minimises the GCV. The only caveat is that by modifying one of the local parameters, one affects the optimal value of the others, and so — as with global regularisation — this has to be achieved in a re-estimation fashion, by optimising each parameter one at a time and repeating until convergence is reached.

Subset selection

As has been discussed, regularisation provides one means for establishing a balance between training data fit and function smoothness. One aspect that was overlooked in regularisation was that each RBF centre was forced to be included in the model — an alternative is to choose some subset of the centres. In practise, considering all 2^m possible subsets is intractable but, fortunately, other heuristics have been found that give good results. Orr [1996] describes a process known as *forward selection*, in which one starts with a model that has no radial basis functions and to which RBFs are added one at a time, at each step choosing the one which most reduces the sum-squared error in the dataset. By monitoring one of the model selection criteria as each RBF is added, one can see which subset minimises the chosen criterion and consequently can be used as the subset of RBFs that define the final model. Orr goes on to show how this may be combined with regularisation, albeit only global regularisation.

Centre and support radius selection

Clearly the choice of support radii for the basis functions plays a large role in the resulting approximant, as differing support radii give rise to different design matrices and thus different solutions for the weights w_i. Similarly, the choice of RBF centres also has a huge impact: in most cases — and indeed, as RBFNs have been presented thus far — the centres are simply chosen to coincide with the training inputs, but this is not a requirement.

Unfortunately, for both of these optimisations there is no known convenient re-estimation formula equivalent to that used for the regularisation parameters, and optimising such features would require the use of a full non-linear optimisation algorithm. Fortunately, there are some less drastic alternatives.

For the case of choosing a suitable support radius, Orr [1999] advocates simply iterating over a set of possible radii and minimising the chosen MSC for each: the radius corresponding to the minimum MSC is then used. Whilst simplistic, it is reasonably effective and fairly efficient when used in

conjunction with the eigenvector and eigenvalue re-estimation method described in the same report. Catering for a support radius that varies across the RBFs is obviously a harder process, so in practise the values iterated over are actually multipliers that are applied to the possibly different radii of the RBFs which are chosen according to some other metric, possibly related to the spatial distribution of the data.

For the problem of centre selection, Orr describes a method that uses *regression trees* [Breiman et al., 1984]. In practise, these are much like *kd*-trees, in that they represent a spatial subdivision of the data using hyper-planes parallel to the axes. In a classical regression tree, the leaf-nodes represent a sub-division of the space in which none of the nodes overlap, and each node stores the average output value of all the training inputs that it contains — for example, Figure 38 shows a curve and its resulting regression tree approximation.

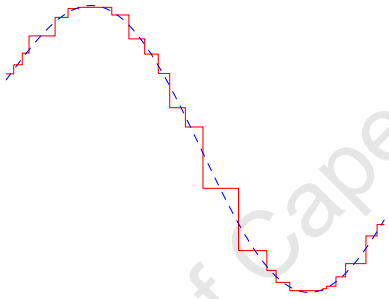


Figure 38: *An example of a function (in blue) and its approximation using a regression tree (in red).*

Standard regression trees have two major problems, namely that they do not adapt well to curvature and that they exhibit discontinuities. Orr shows how regression trees can rather be used for choosing the centres and support radii of RBFs, which then provide a smoother and continuous fit to the data. His technique also lends a multi-scale aspect to the solution of function approximation, as he considers all nodes in the regression tree (right from the root down to the leaves) when creating potential RBFs — meaning that at the root there is an RBF which covers the entire input space, and successive levels down the tree refine the approximation by covering smaller spaces, naturally adapting according to data density and data clustering. Having generated a set of centres and support radii for the RBFs, regular techniques such as forward selection and regularisation can be applied.

An additional technique for selecting centres is possible, which takes into account the multi-scale structure of the tree. Starting from the root and working down through the tree, RBFs are considered for addition to the network based on how their inclusion would affect the MSC. The tree structure also affords the ability to analyse relationships between RBFs better, and observe how the MSC would be impacted by, say, removing a parent and rather including its two children.

Ohtake et al. [2004] take a different approach. They define an error metric for a given support radius and center, and then describe a means for optimising the choice of the support radius. Using this, they go on to select a subset of the input points as the RBF centres in a randomised greedy fashion,

based on how little support is provided to each of the points by the current set of centres.

Samozino et al. [2006] extend the notion of centre selection to include points not in the original input. By exploiting the fact that the function they wish to approximate is the *zero-level set* of a manifold surface, they approximate the medial axis of the surface by using a Voronoi tessellation of the input points. Choosing what are known as *poles* from the vertices of the Voronoi diagram, gives a set of points close to the medial axis, as shown by Amenta et al. [2001] and Boissonnat and Cazals [2000]. Samozino et al. then go on to choose their centres as a subset of the set of poles, using k-means clustering [MacQueen, 1967] as a guide to selecting the specific subset of poles to use. As the resulting centres are close to the medial axis, it is thus easy to extract suitable support radii for the RBFs as the distance from each RBF to the point set surface.

Other RBFN considerations

RBFNs are typically very good at approximating finer details in approximations, but perform less well if there are also large global features. RBFNs are also local by virtue of their support radii, and so suffer when required to exapolate. A simple means for ameliorating such a situation is to augment the radial basis function network with an underlying polynomial approximation, giving a function of the form

$$f(\mathbf{p}) = \sum_{i=1}^{m} [w_i h_i(\mathbf{p})] + P_m(\mathbf{p})$$

where $P_m(\mathbf{p}) = \mathbf{b}(\mathbf{p}) \cdot \mathbf{c}$ is a polynomial in $\prod_m^d$ with bases $\mathbf{b}$, and coefficients $\mathbf{c}$ that are found using a least squares approach as described in Section 4.2.1[5]. The benefit of such a formulation is that it offers both local and global properties — the polynomial provides a global approximation which allows for extrapolation to areas not covered by the support radius of radial basis functions, whilst the RBFN serves as a good local approximant in more dense areas of data. Care must be taken to not overfit the data, however, and so either a low-order polynomial must be used or some human assessment should be done to decide on the best order to use.

4.3 Online learning

The approximation techniques discussed thus far have all made use of a fixed training set. Online learning techniques differ in this regard, in that they are able to incrementally adjust the resulting approximant as more training data is provided.

[5]Typically $m = 1$ giving the best hyperplane fit through the data.

4.3.1 Artificial neural networks

The inception of research into *artificial neural networks* is widely regarded as being due to McCulloch and Pitts [1943], and later expanded by Kleene [1956]. The fundamental principle behind this work is to accurately model the workings of biological *neurons*, and led to the development of the *artificial neuron* which receives one or more inputs and sums these to generate an output. In typical applications the inputs to the sum are weighted, and the sum is also transformed by a non-linear function known as the *activation function*.

More formally, an artificial neuron i can be in one of two states: $x_i = 1$ (firing) or $x_i = 0$ (not firing). Neurons are connected in the same way that synapses connect biological neurons, where the weight associated with each connection indicates the strength and type of connection (positive weights indicate excitatory relationships whilst negative weights indicate inhibitory relationships). For a neuron to become active, the sum of the weights of active connected neurons, combined with a per-neuron bias, must be greater than zero. This is represented mathematically by:

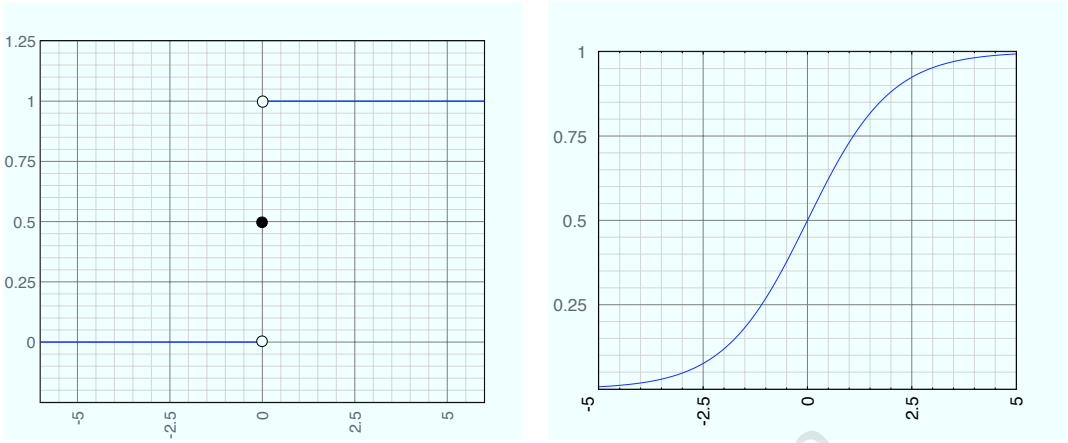
$$x_i = \theta(b_i + z_i) \tag{7}$$

$$z_i = \sum_{j \in C_i} w_{ij} x_j \tag{8}$$

where w_{ij} gives the weight of the connection between neurons i and j, C_i is the set of neurons connecting to neuron i and b_i is the bias for neuron i. θ is the previously mentioned activation function — which for binary neuron states is most often chosen to be the *Heaviside step function*[6]. Artificial neurons that are modelled on these mathematical principles are more commonly referred to as *threshold linear units*, *threshold logic units*, or more simply, TLUs. More complex extensions to this basic model for a TLU are available, the most common of which are the addition of a time index to the x_i states (giving $x_i[t]$ which depends on the values $x_j[t-1]$ of its neighbouring neurons) and the relaxation of the output of x_i to a real value in the range $[0; 1]$. The latter is employed particularly often in conjunction with a *sigmoid activation function* (see Figure 39(b)), as the output from the TLU then varies continuously with respect to its inputs — an oft desired characteristic for functions. The sigmoid function also has the property that $\frac{d\theta}{dx} = \theta(1 - \theta)$ which, as we will later show, is extremely useful in selecting the weights w_{ij}.

It should be readily apparent that a single TLU is not particularly powerful: suppose we had a TLU that took as input the output of two other neurons (in other words, a TLU that models a function of two boolean variables whose output is a single boolean output). Now it is impossible for this TLU to represent the *exclusive-or* of its two inputs [Minsky and Papert, 1969] — Table 3 shows the various constraints that each input/output pair imposes on the weights and bias of the TLU if one supposes that it is possible to use a single-TLU representation.

[6] The step function outputs 1 when it's argument is positive, and 0 otherwise: see Figure 39(a).



(a) The Heaviside step function defined by $\theta(x) = 0$ for $x \leq 0$ and $\theta(x) = 1$ for $x > 0$

(b) The so-called *logistic function* — a member of the sigmoid family of functions — defined by $\theta(x) = \frac{1}{1+e^{-x}}$

Figure 39: *Examples of activation functions used in neural networks.*

x_1	x_2	*output*	Constraints imposed
0	0	0	$b \leq 0$
0	1	1	$b + w_2 > 0$
1	0	1	$b + w_1 > 0$
1	1	0	$b + w_1 + w_2 \leq 0$

Table 3: *The possible inputs and corresponding outputs for the exclusive-or function, and the associated constraints these infer on the parameters of a TLU that would model such a function.*

From the first two constraints in Table 3, it follows that $-b \geq 0$ and $w_2 > -b$, which together give $w_2 > 0$. Then $b + w_1 + w_2 = (b + w_1) + w_2 > 0$ from a combination of the third constraint and the derivation of $w_2 > 0$. This directly contradicts the fourth constraint, however, and so the assumption that the exclusive-or function can be modelled by a single TLU must be false.

How then are TLUs useful? The trick is to combine several of them together in such a way that their outputs are directed as inputs to other TLUs – Figure 41 shows a diagram of a *multilayer perceptron* [Haykin, 1994] which is the most commonly used form of this generalisation. Pertaining specifically to the discussion of the exlusive-or function above, the contradiction arose when an attempt was made to satisfy the final constraint: namely when both the inputs to the function are *true*. Figure 40 shows how this is addressed by adding a second TLU to handle that specific constraint.

Multilayer perceptrons are, in fact, extremely powerful. Hornik et al. [1989] proved that a multilayer perceptron with a single hidden layer is capable of approximating any continuous function that maps an interval of real numbers to some output interval of real numbers, using a finite number of artificial neurons in the hidden layer. This is known as the *universal approximation theorem*, and can be generalised to functions in any finite number of dimensions. With this theoretical justification for the usefulness of a neural network, what remains are the tasks of choosing the network structure (the number of hidden nodes and the connectivity information) and determining suitable weights.

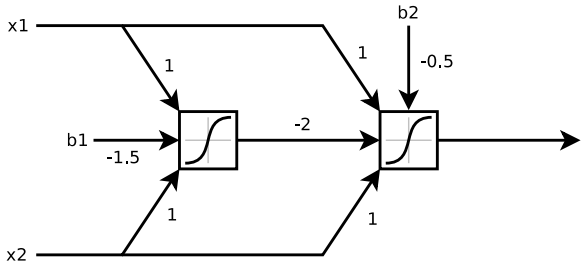


Figure 40: *The exclusive-or function of two variables, modelled using two threshold logic units.*

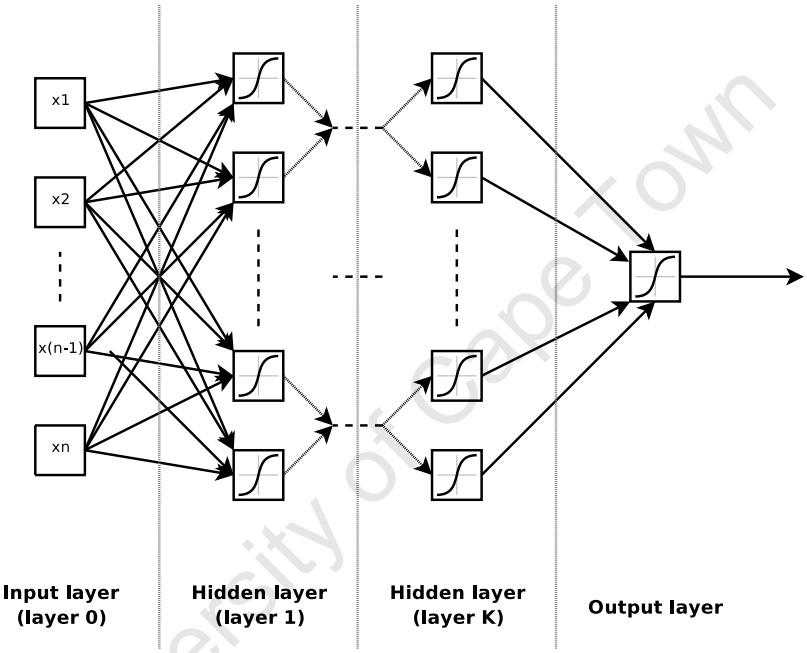


Figure 41: *A schematic for a multilayer perceptron. The artificial neurons are organised into layers, consisting of the input layer, output layer, and one or more hidden layers. Each neuron in any of the hidden layers receives its input from the previous layer, and passes its output to the next layer.*

Deciding the number of hidden nodes is unfortunately a non-trivial task, and a problem for which no general solution currently exists. There are a number of different heuristics that have been employed, although these are typically derived for use with specific datasets where empirical testing has shown which configurations work best.

One might question why this poses such a challenge. It should be fairly intuitive to see that with too few nodes, a network may not be able to capture all of the information required (as a simple example, consider a hidden layer with only one node — this is in fact no better than a single TLU). Then surely the easiest solution is to have many nodes — an ample supply for the amount of information we wish to capture. The trouble is that this is also problematic, and leads to overfitting — a condition where the function is able to approximate with great accuracy the data in its training corpus, but generalisations to other inputs give extremely poor results. A simple example of this concept, with regard to the related task of fitting a polynomial to data, would be if one were trying

to fit data that has been sampled from a straight line with a higher order curve (see Figure 42).

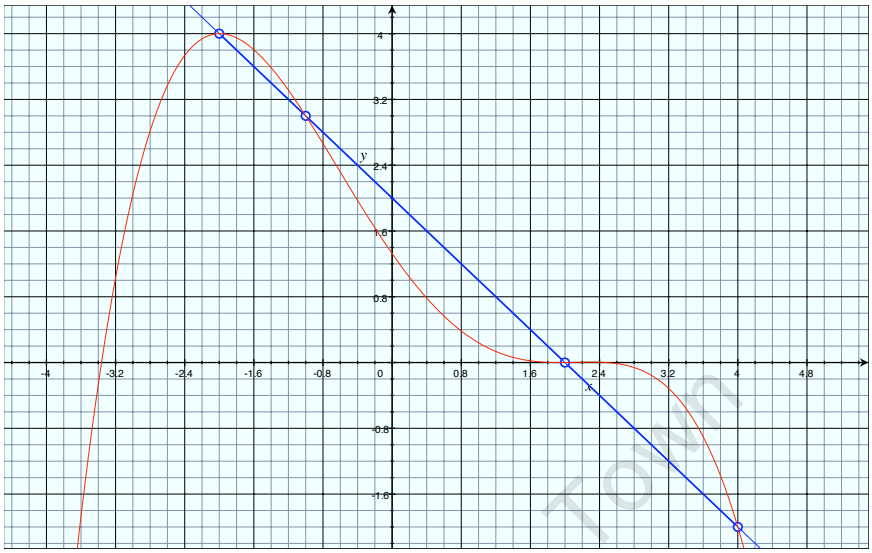


Figure 42: *An example of how overfitting can occur. In this example, four points are sampled from the straight line defined by* $y = -x + 2$ *(shown by the blue line). If we had no prior knowledge of the nature of the data, we might try to fit the data using a quartic, giving the red curve which would clearly give poor results for other data samples — especially for* $x < -2$ *where the red curve has positive gradient and the blue curve has negative gradient, causing them to diverge further apart as* $x \to -\infty$*.*

Although there is no consensus amongst researchers for choosing a suitable number of hidden units, it is generally agreed that the number of inputs, number of outputs and number of training samples all play a part and should be taken into account when deciding how many hidden units to use. A commonly advocated strategy is to keep aside some training data as testing data, which can then be compared against the network's output for a number of configurations so as to determine which configuration seems to perform the best.

With regard to the connectivity of the network: since most applications tend to use only a single hidden layer as a result of the universal approximation theorem, it makes little sense for the connections between the hidden layer and the output TLU to be anything but densely connected (that is, every node in the hidden layer feeds its output forward into the output TLU). Thus the only part of the network worth really dwelling on is the connectivity between the input layer and the hidden layer. Most applications tend to simply employ dense connection here as well: that is, every input node is connected to every hidden node. This generally works well, as any connection which corresponds to a mismatched correlation will typically end up with a weight close to 0 during the process of training the network. The only practical case where it may be necessary to prune the connectivity of the network would be when one has specific knowledge of the problem that pertains to the separation of the input nodes into specific, distinct groups.

Once a network configuration has been decided on, what remains is to assign the best possible weights to the edges of the network. The technique of *backpropagation* is the most widely used, and

was first described by Werbos [1974]. The key idea behind backpropagation is to feed an input into the network for which the output is known, and then use the error in output to adjust the weights of the network from back to front — propagating the errors back until the input layer is reached (hence the name given to the technique). This process is repeated with all the inputs from the training set, and then again either for a fixed number of iterations or until some external error criterion has been satisfied[7].

More formally, at each step backpropagation seeks to minimise the error E of the network output by calculating $\frac{\partial E}{\partial w_{ij}}$, so that when the error is minimised $\frac{\partial E}{\partial w_{ij}} = 0$. These partial derivatives also provide a means for indicating how each weight should be adjusted: by using the approach of gradient descent, backpropagation gradually alters the weights so that the partial derivatives approach 0 as the training continues. Suppose an input is fed into the network, with expected output o_k from the output node k. The error of the network for this is expressed in the mean squared sense as

$$E = \frac{1}{2}[o_k - \theta(b_k + z_k)]^2$$

where the constant $\frac{1}{2}$ is used to simplify the derivation in the next few steps: it essentially becomes absorbed in the learning parameter η, discussed later. The partial derivative of E with respect to weight w_{kj} can then be derived as

$$\begin{aligned} \frac{\partial E}{\partial w_{kj}} &= \frac{\partial E}{\partial z_k} \cdot \frac{\partial z_k}{\partial w_{kj}} \\ &= \frac{\partial E}{\partial z_k} x_j \end{aligned}$$

as from Equation 8 we have $z_k = \sum_{j \in C_k} w_{kj} x_j$. Since $\frac{\partial E}{\partial z_k}$ is independent of j, one can denote this by δ_k and derive

$$\begin{aligned} \delta_k = \frac{\partial E}{\partial z_k} &= \frac{\partial}{\partial z_k} \frac{1}{2}[o_k - \theta(b_k + z_k)]^2 \\ &= -(o_k - x_k) \frac{\partial \theta(b_k + z_k)}{\partial z_k} \end{aligned}$$

To continue, θ must be differentiable — this is why the choice of the sigmoid for the activation function is so useful, and since $\theta' = (1 - \theta)\theta$, no complicated derivative computations are required. Now

[7] Typically, a data set is divided into three parts — a training set, a testing set and a validation set. The testing set is used to ensure that the network generalises sufficiently, and allows one to try several different network configurations until a suitable network is determined. A final run is then done against the validation data, which has thus far not been used for any of the network optimisation. This ensures that the network is sufficiently general.

$$\begin{aligned}\delta_k &= -(o_k - x_k)\frac{\partial\theta(b_k + z_k)}{\partial z_k} \\ &= -(o_k - x_k)(1 - \theta(b_k + z_k))\theta(b_k + z_k) \\ &= -(o_k - x_k)(1 - x_k)x_k\end{aligned}$$

Employing gradient descent, the change for each weight w_{kj} can now be written as

$$\Delta w_{kj} = -\eta\frac{\partial E}{\partial w_{kj}} = -\eta\delta_k x_j \tag{9}$$

where η is the *learning rate* of the network, which affects the rate with which weights may change. Large vales of η could cause the weights to oscillate without ever converging, whilst very small values of η could require too many training samples in order to achieve convergence. The learning rate is typically fixed (at, for example, a value of 0.05) although some work has been done toward dynamically adjusting the learning rate [Yu and Chen, 1997].

Note, however, that Equation 9 was derived specifically for the output node from the network, and for the weights of the edges that are inputs to the output node. The error still needs to be propagated to the remaining nodes and their input edges' weights. For the weight w_{ij} of an edge that is an input to hidden node i, one can make judicious use of the chain rule to write

$$\begin{aligned}\frac{\partial E}{\partial w_{ij}} &= \frac{\partial E}{\partial z_k}\cdot\frac{\partial z_k}{\partial x_i}\cdot\frac{\partial x_i}{\partial z_i}\cdot\frac{\partial z_i}{\partial w_{ij}} \\ &= \delta_k w_{ki}\frac{\partial\theta(b_i + z_i)}{\partial z_i}x_j \\ &= \delta_k w_{ki}(1 - x_i)x_i x_j\end{aligned}$$

and similarly the change for each weight w_{ij} is

$$\Delta w_{ij} = -\eta\delta_i x_j$$

where

$$\delta_i = \delta_k w_{ki}(1 - x_i)x_i$$

Having computed the change for all the weights in the network, each weight w_{ij} is then updated by setting $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$.

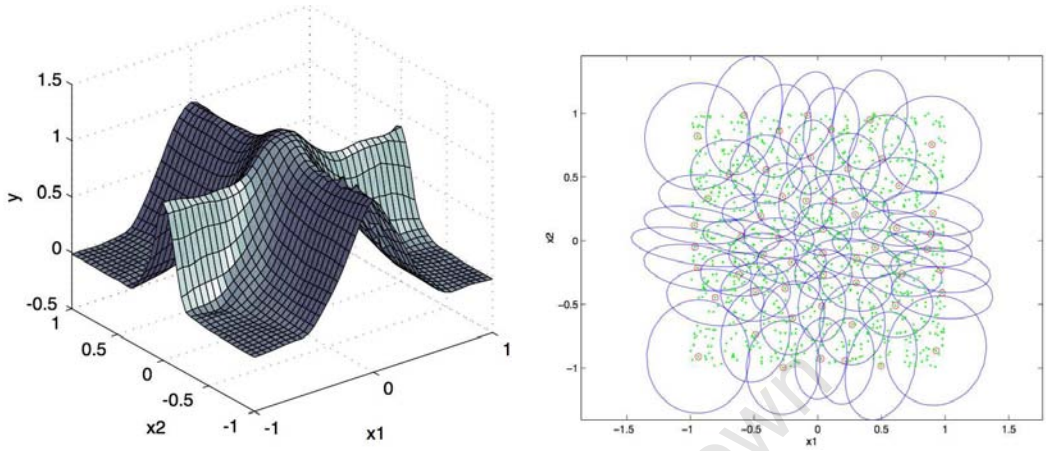
With a mathematically sound method for learning the weights in a neural network, it is worth considering their extensibility to high dimensions and contrasting them to other techniques. The space and time complexities of neural networks are strongly coupled to the number of nodes in the network. As has been discussed, this is not a fixed number and can vary significantly, but in general the total number of nodes is linearly proportional to the number of input dimensions and, by virtue of the fact that one has control over the number of hidden nodes, the overall complexity can be adjusted by adding or removing these nodes. As such, neural networks are well suited to function approximation in high dimensions.

In contrast to many of the techniques that have been presented thus far, neural networks stand apart in the sense that they model the problem of function approximation in a bottom-up fashion. Where other techniques seek to establish some higher-level understanding of the nature of the function being approximated, neural networks instead use a large number of simple components to achieve complexity. For this reason, neural networks can be extremely difficult to analyse, as for even a modest network it is not at all clear exactly how the network operates or what high-level concepts — if any — each of the hidden nodes represents. With the wrong learning rate or number of hidden nodes, neural networks are also known to map the inputs with which they are trained extremely well, but perform poorly on any other inputs — that is, to establish a neural form of lookup table. These caveats can make neural networks extremely difficult to setup, but once correctly structured and trained these networks can perform extremely well.

4.3.2 Locally weighted projection regression

A more recent technique compared with those discussed so far is that of Vijayakumar et al. [2005], which specifically addresses the issue of function approximation in high-dimensional spaces. They observe that high-dimensional data collected through physical observations often has much lower true dimensionality — that is, there are strong correlations between many of the dimensions and consequently the actual data is not as complex as its containing high-dimensional space makes it appear. Their technique, *locally weighted projection regression* (LWPR), makes use of this fact to reduce the dimensionality of the data, and then employs existing linear models that perform well on smaller numbers of dimensions to locally approximate the overall function. Similarly to neural networks, the LWPR system is trained incrementally by processing one training point at a time and making adjustments accordingly. Initially the model has no local functions, and as training proceeds new local functions are added as needed, based on the spatial locality and density of the training points processed. This allows for LWPR to capture only those small parts of the high-dimensional space in which the data is clustered — which means that in these areas it will approximate well, but will fail to generalise if the true space occupied by the data is not well sampled. Figure 43 shows an example of a synthetic function to be approximated and the resulting placement of local models in

response to the training data.



(a) (b)

Figure 43: *An example of (a) a synthetic cross function and (b) the resulting placement and area of support of local models (shown in red and blue) in response to the training points (shown in green). As can be seen, the corners of the function exhibit lower magnitudes of gradient and so can be well approximated with only a single local model, whilst the areas with larger magnitudes of gradient require more local models to capture the nature of the function.*

4.4 Summary

In this chapter, a variety of techniques have been presented that address the problem of function approximation from a given set of training samples. The following broad categories were considered:

1. **Scattered data interpolation** techniques produce curves that pass exactly through each of the points provided in the training set. These interpolants are susceptible to outliers, and in most cases are only suitable for mapping between very small numbers of dimensions. Interpolation techniques are also not well-suited to generalisation, unless an independent source of gradient information is available.

2. **Scattered data approximation** methods differ from the interpolation techniques in that they do not constrain the resulting approximant to pass exactly through each training point. Instead, a better shaped curve is sought at the expense of an imperfect interpolation. In most cases the derivation of these curves involves the solution of a least squares system, which may be significantly costly depending on the number of dimensions and the degree of the underlying polynomials being fitted. Scattered approximation techniques typically achieve good approximations and are able to extrapolate easily.

3. **Online learning** techniques differ in that they allow for the training data to be incrementally altered, without having to completely repeat the training process with the new data. This has the advantage that an approximant can be refined as it is being used.

In the following chapter, characteristics that are required for the specific application of function approximation to a mapping between adjective space and parameter space are discussed, and full details of the technique are presented.

Chapter 5

System architecture

In Chapter 4, the general problem of function approximation was explored by considering a number of techniques that are typically used. Some caveats, such as problems with higher dimensions, have already been discussed, but before choosing a function approximation technique to apply to adjective space, there are some additional issues that need to be addressed. These include:

1. **Direction of mapping:** whilst a function of the form $f : \mathcal{A} \rightarrow \mathcal{P}$ is desired, it is shown that, for several reasons, the function that should really be approximated is $g = f^{-1} : \mathcal{P} \rightarrow \mathcal{A}$, and solutions for dealing with the inverse function are discussed and derived.

2. **Adjective representation:** whilst the general concept of using adjectives sounds good, the technical details regarding their representation and consequent implementation have not yet been discussed. These issues are thoroughly explored in this chapter.

3. **Support for common use-cases:** the formulation of adjective space thus far makes it seem that

   - The set of adjectives available to users is fixed.
   - When a user wishes to make a description, they have to specify a *point* in adjective space, which implies that they have to choose a value for *every* dimension/adjective.

   Both of these limitations are discussed, and taken into consideration for the final system design.

Through addressing these problems, an overall system architecture will emerge. We divide the system into two major constituents — a *modelling* component, and a *usage* component. In the modelling component, a procedural system is used to generate random content samples, which are described by an expert user or designer. These descriptions are used to train a set of specialised radial basis function networks, which together comprise the adjectival model. In the usage component, a user describes the content that they wish to generate using adjectives. These are then processed by the

model to obtain suitable procedural parameters, which are then fed into the procedural system to obtain content. This process is illustrated in Figure 44.

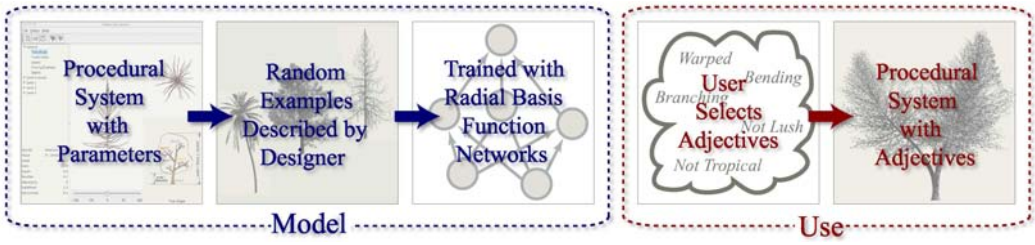


Figure 44: *An illustration of the overall system proposed by this thesis.*

5.1 Direction of mapping

The data collection process described at the beginning of Chapter 4 operates under the principles of a classic experiment with random assignment [McBurney and White, 2006]. As such, the data acquired actually models the relationship from parameter space to adjective space, since the user provides descriptions in adjective space in response to content generated by a point in parameter space. Were this data used to approximate a function it would be $f^{-1} : \mathcal{P} \to \mathcal{A}$, the inverse of the function that is sought. Furthermore, performing data collection so as to achieve the correct causal relationship is infeasible: that would amount to giving the user a description and having them find the procedural parameters matching this description — exactly the problem that adjective space seeks to address.

Additionally, Chapter 3 highlighted that the function $f : \mathcal{A} \to \mathcal{P}$ could conceivably not be onto, or could be a one-to-many function — neither of which are desirable properties in the specific context of mapping between adjective space and parameter space. It was demonstrated that the inverse mapping overcomes these problems — setting the domain of the function to be $\mathcal{P}$ guarantees that every piece of content can be associated with some point in $\mathcal{A}$, and multiple points in $\mathcal{P}$ can share the same description in $\mathcal{A}$. It is worth noting that f^{-1} is also not necessarily onto, but this is acceptable as one could conceive of descriptions in adjective space that will not apply to any generated content.

The trick, now, is that what is *really* sought is a way of mapping from $\mathcal{A}$ to $\mathcal{P}$. Suppose the user describes a scene they wish to generate, by choosing $\mathbf{a} \in \mathcal{A}$, and a suitable approximation to f^{-1} has been determined using some function approximation technique. What is then required is to solve the equation $f^{-1}(\mathbf{p}) = \mathbf{a}$.

Whilst this is, in general, a hard problem, it does afford a useful benefit. Since f^{-1} could possibly be a many-to-one function, there may be multiple points in $\mathcal{P}$ that satisfy the above equation — this allows for more than one solution to be found and presented to the user, in contrast to the function f which would map to exactly one (possibly incorrect) point in $\mathcal{P}$.

In solving the equation, it is important to remember that f^{-1} is not necessarily onto. As such, the conditions on the equation must be relaxed slightly to instead give $f^{-1}(\mathbf{p}) \approx \mathbf{a}$. Exactly how one quantifies what constitutes the best approximating solution to this equation is open to debate, but the most common approach is to minimise the squared difference $E(\mathbf{p}) = \|f^{-1}(\mathbf{p}) - \mathbf{a}\|^2$.

Dependent on the properties of the objective function E, certain avenues are available for its minimisation. If, for example, E is differentiable, then one could employ a technique such as the Newton-Raphson method [Newton, 1664–1671; Raphson, 1690; Kelley, 2003] to find local minima. If E is differentiable and it is easy to determine the roots of E, then one can easily extract local minima without the need for complex root-finding algorithms.

In general, since E is the combination of some potentially complicated function f^{-1} and a squared error computation, it is unlikely that simple techniques will suffice. There is, however, still a solution: the use of space-searching techniques such as *genetic algorithms* (GA) [Holland, 1975; Goldberg, 1989] or *particle swarm optimisation* (PSO) [Kennedy and Eberhart, 2001]. Both of these methods make use of a population of possible solutions, which is then iteratively updated over a large number of steps in an attempt to find the solution which minimises E.

5.1.1 Genetic algorithms

Neural networks seek to model biological processes. Similarly, genetic algorithms were born from the field of evolutionary biology and seek to mimic Darwin's process of *natural selection* [Darwin, 1859].

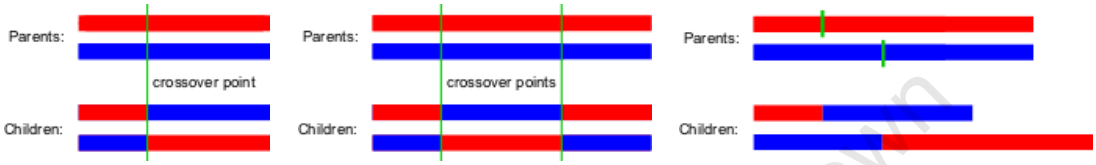
As has already been mentioned, genetic algorithms operate on a population of solutions to a problem. To be more precise, the population is actually a set of *abstract representations* of the real solutions, and each member of the population is referred to as a *genotype* whilst the actual solution derived from a genotype is called the *phenotype*. Starting from a randomly generated population of individuals, the next generation of the population is determined by first computing the *fitness* of each individual — which in our context is measured by the error E. The next generation is then generated by a stochastic selection and modification of the current population, and this is then used to similarly create the following generation. The process continues until either a certain number of generations have been produced, or until the fitness of the best individual exceeds a set target.

The relation to Darwin's natural selection comes into play in the abstract representation of the population's individuals, and when populating a generation's successor. The most common representation for an individual is as a simple series of bits — drawing a parallel to chromosomes which are made up of nucelotide bases. Other popular representations include arrays of real values or tree-like structures, but these require some extra care when defining genetic operators that function similarly to those used on bit series.

Once the fitness of each individual has been computed, a subset of the population are chosen to

"breed" the following generation. This is done stochastically, in such a way that individuals with higher fitness are more likely to be selected. Having selected a set of progenitors, these are manipulated in one of three ways to produce individuals for the next generation:

1. **Crossover.** Just as in nature where the offspring of a species are often composed of the genetic material from two parents, so too can new individuals be created in the simulated population. For an abstract representation as an array of bits, there are three commonly used methods for applying crossover, which are shown in Figure 45.



(a) One-point crossover (b) Two-point crossover (c) "Cut and splice" crossover

Figure 45: *Methods for crossover in genetic algorithms.*

2. **Mutation.** Related to the phenomenon of biological mutation, mutation on a series of bits is simply determined by generating a random variable for every bit. If the value of the variable is less than a pre-specified threshold, then that bit has its state flipped. This allows a genetic algorithm to maintain diversity and avoid local minima by ensuring that all individuals do not become too similar to each other.

3. **Elitist selection.** Some of the better individuals from the current population are carried through to the next generation without any modification.

One of the shortfalls of genetic algorithms is that emphasis is placed on the importance and fitness of individuals, as opposed to the fitness of the population as a whole. This is also apparent in the way in which successive generations are produced, in that each new individual is developed from the abstract representations of only two parent individuals. *Gaussian adaptation* [Kjellström, 1991] is a closely related algorithm that seeks to maximise the mean fitness of the whole population, provided that the ontogeny of an individual can be described as a sequence of small, random evolutionary steps, and that the resulting phenotype tends to be Gaussian distributed. For domains that satisfy these restrictions, this allows the optimisation process to follow ridges in the phenotypic landscape, which can lead to increased performance of several orders of magnitude. These restrictions do, however, limit the applicability of the technique and so it has not yet gained widespread adoption in the scientific community.

5.1.2 Particle swarm optimisation

Particle swarm optimisation is an effort to overcome the problems with genetic algorithms discussed above. As before, one has a population of individuals — referred to as a *swarm* — but in this case

each individual is modelled as a particle in a multidimensional space. Each of the particles has position and velocity within this space, and these are adjusted at discrete time-steps according to reasoning made by the particle regarding both the particle's best observed position, as well as the swarm's overall best observed position. In this way, there is global knowledge shared amongst all of the individuals as well as localised knowledge particular to each individual — much as a swarm of insects would function, which is the basis for the rationale behind this technique.

Formally, suppose we have a swarm of particles with position and velocity given by $\mathbf{x_i}, \mathbf{v_i} \in \mathbf{R}^{|\mathcal{P}|}$, and let $\mathbf{b_i}$ be the best observed position by particle i and $\mathbf{g}$ be the global best position observed by any member of the swarm. Particle swarm optimisation then proceeds as follows:

- **Initialisation.** The positions $\mathbf{x_i}$ and velocities $\mathbf{v_i}$ are initialised: typically, the $\mathbf{x_i}$ values are chosen so as to be uniformly distributed over the input domain and $\mathbf{v_i} = \mathbf{0}$. The values $\mathbf{b_i}$ are initialised as $\mathbf{b_i} \leftarrow \mathbf{x_i}$ and $\mathbf{g} \leftarrow \min_{\mathbf{x_i}}\{E(\mathbf{x_i})\}$.
- **Iterative update step.** For each particle, the following update steps are followed:
  - **Update position.** This simply entails moving the position of the particle according to it's current velocity: $\mathbf{x_i} \leftarrow \mathbf{x_i} + \mathbf{v_i}$
  - **Update velocity.** Using the knowledge of its own best observed position and the best position observed by the swarm, the particle updates its velocity so as to move towards these best positions. This is achieved by setting $\mathbf{v_i} \leftarrow \omega\mathbf{v_i} + c_1\mathbf{r_1} \circ (\mathbf{b_i} - \mathbf{x_i}) + c_2\mathbf{r_2} \circ (\mathbf{g} - \mathbf{x_i})$.[1] ω is an inertial constant which is typically slightly less than 1. The constants c_1 and c_2 control how rapidly a particle should adjust its velocity so as to set a course for the good positions it knows about — these are said to represent "cognitive" and "social" aspects, respectively, of the particle's decisions. Finally $\mathbf{r_1}$ and $\mathbf{r_2}$ are vectors of uniformly sampled random values between 0 and 1 that allow the particles to exhibit stochastic exploration of the search space.
  - **Best position updates.** Before iterating the next particle, the current particle's new position is checked to see if it is better than either its personal best or the global best: if $E(\mathbf{x_i}) < E(\mathbf{b_i})$ then $\mathbf{b_i} \leftarrow \mathbf{x_i}$; if $E(\mathbf{x_i}) < E(\mathbf{g})$ then $\mathbf{g} \leftarrow \mathbf{x_i}$.
- **Check for convergence.** Similarly to genetic algorithms, the previous step is repeated either a fixed number of times or until a suitable global best has been achieved.

A simple and oft-used extension to the standard PSO algorithm is to initially divide the swarm into a set of overlapping neighbourhoods based on the index of the individuals (for example a neighbourhood containing particles 1 to 5, one containing particles 2 to 6, etc). Instead of maintaining a global best for the entire swarm, one now keeps track of a best position for every neighbourhood — and when a particle discovers a new best position it only updates that for the neighbourhoods

[1] $\circ$ in this context indicates the componentwise multiplication of the corresponding vector elements to form a new vector, as opposed to the dot product. That is, if $\mathbf{a} = \mathbf{b} \circ \mathbf{c}$ then $a_i = b_i c_i$.

of which it is a member. This allows for a better exploration of the search-space and can greatly reduce the chance that a PSO will become susceptible to a local minimum, with the caveat of slower convergence to the best solution. Because of the overlapping nature of the neighbourhoods, the global best will eventually propagate through all the neighbourhoods, and so one still maintains the benefits of a swarm intelligence approach over techniques such as GAs.

5.1.3 General comments on genetic algorithms and particle swarm optimisation

Non-linear search algorithms, such as GAs and PSOs, have the following three advantages:

1. **Global search**. Searching is not restricted to some locality around which a solution is expected (which is required for Newton-Raphson); the entire solution space is considered as possible candidates for the optimal solution. Priming a GA or PSO with a possible solution would in fact bias the methods towards a minimum in that area, which could possibly result in a local minimum being found but never truly escaped.
2. **Avoidance of local minima**. Having said that these techniques can become stuck in local minima, both GAs and PSOs actually have mechanisms for considering solutions other than the best minimum they have found thus far by randomly examining other areas of solution space. In so doing, local minima are usually avoided, which is where techniques such as Newton-Raphson fail. The escape from a local minimum is not guaranteed, however, and is more likely to work when a broader sampling of the solution space is initially specified (that is, not primed towards a particular mimimum).
3. **Not property dependent**. GA's and PSO's are not dependent on the properties of E.

Although these techniques look extremely useful, they have one drawback: they never terminate conclusively. GA's and PSO's can never actually guarantee that they have found *the best* solution to an equation; they can only provide continuous feedback on the best solution that they have found thus far. As has been discussed, there is usually some form of user-defined threshold ϵ, such that once $E(\mathbf{p}) < \epsilon$ the procedure terminates; or more simply a fixed number of iterations are run after which the process is discontinued. Alternatively, the procedure could be allowed to run *ad infinitum* until the user decides that the calculated error is tolerable.

This is not a catastrophic problem, however. If anything, this allows for several possibilities within the framework of adjective space:

1. Whenever the GA or PSO finds what it deems to be a local minimum, this could be presented to the user as a possible output. The user would then have the option of accepting the generated content, or rejecting it and effectively forcing the GA or PSO to keep searching.

2. The GA or PSO is run in the background, and periodically updates the user interface with its best result. This allows the content available to the user to change dynamically, as the search discovers alternative options. Through observing pieces of generated content, the user would then have the option at any stage of deeming a particular piece of content suitable, thus terminating the search, or leaving the system to keep running and searching for suitable options.

In our technique, we have chosen to use particle swarm optimisation — the implementation of the PSO algorithm is much simpler than that of a GA, and also more naturally adapts to the real-valued parameter space that we employ in our experimentation[2]. We have recorded successful results using particle swarm optimisation, and so we have not explored the use of genetic algorithms. However, we expect that similar results could be achieved with genetic algorithms — and indeed, for procedural systems whose parameters are more naturally altered by the genetic operators, genetic algorithms may prove to be superior. The *no free lunch theorem* [Wolpert and Macready, 1995, 1997] also guarantees that particle swarm optimisation will not be suitable for all scenarios, and in such cases genetic algorithms may prove more suitable due to the different manner in which they explore the search space.

To summarise, having identified that a problem exists in approximating a mapping from $\mathcal{A}$ to $\mathcal{P}$, it has been shown that approximating the inverse mapping is feasible and also offers additional benefits (another of which is discussed in more detail in Section 5.4). For brevity and clarity in the remainder of this chapter, it will be assumed that g refers to the inverse mapping $\mathcal{P} \rightarrow \mathcal{A}$, except where a distinction between the inverse and regular mappings needs to be made, in which case it will be explicitly stated.

5.2 Properties of g

The various function approximation techniques discussed in Chapter 4 have a number of different characteristics, and hence are applicable to different problems. In order to choose a method that works well in mapping between parameter space and adjective space, it would be prudent to consider desriable properties of the resulting function.

5.2.1 Continuity

Suppose a point $\mathbf{p} \in \mathcal{P}$ is mapped to a point $\mathbf{a} \in \mathcal{A}$ by the function g. Consider now the point $\mathbf{p}' = \mathbf{p} + \epsilon$, where ϵ is some non-zero vector with very small magnitude, and what $\mathbf{p}'$ might be mapped to by g. Logically, $\mathbf{p}'$ corresponds to a set of procedural parameters that is only slightly

[2]Note that this does not preclude the use of genetic algorithms for real-valued search domains, as there are means of encoding real values in bit-strings that can be manipulated by the genetic operators — see, for example, Schraudolph and Belew [1992].

different to $\mathbf{p}$, and therefore one would expect that the content generated by $\mathbf{p}'$ should be reasonably similar to the content generated by $\mathbf{p}$. Consequently, the two pieces of content generated should also elicit similar descriptions — and so one would expect the point $\mathbf{a}'$, to which $\mathbf{p}'$ is mapped, to be similar to $\mathbf{a}$. More technically, one would expect that $\mathbf{a}' = \mathbf{a} + \epsilon'$ for some non-zero ϵ' with small magnitude.

More formally, we might say that

$$\forall \epsilon' > 0, \exists \epsilon > 0 \text{ such that if } \|\mathbf{x} - \mathbf{p}\| < \epsilon, \text{ then } \|g(\mathbf{x}) - g(\mathbf{p})\| < \epsilon'$$

which is simply the Cauchy definition of continuity at the point $\mathbf{p}$. In other words, g is required to be *continuous*.

5.2.2 Generalisation

Suppose one were to construct the convex hull in parameter space around the set of training points, $\mathcal{T}$. The volume enclosed by the hull represents the set of points that could be reasonably approximated using *interpolation* techniques, owing to the fact that any point inside the hull can be expressed in barycentric coordinates as a combination of the vertices contained by the hull. Points which fall outside the hull, however, cannot be expressed in barycentric coordinates and so cannot be dealt with in the same way — that is, the output of the function at these points cannot be calculated using interpolation. Instead, the function must be able to *extrapolate* or *generalise* such points. Giving accurate approximation results at such points is clearly harder since there is no gradient information between the boundaries of $\mathcal{P}$ and the convex hull, but even a coarse approximation would nevertheless be more valuable than no approximation at all.

Due to the *curse of dimensionality* [Bellman, 1961; Scott, 1992], densely sampling higher-dimensional spaces becomes increasingly hard and so we cannot rely on an adequate sampling to avoid the convex hull problem. A technique which is able to extrapolate beyond the hull of its training points would thus be extremely useful.

5.2.3 Locality

When using real-world data, one always has to proceed under the assumption that the data is affected by some form of noise. In the function approximation context, this refers to finding the best approximation to the data that *does not necessarily* interpolate the data — where "best" refers to maintaining some form of $C^{(n)}$ continuity or particular curve shape. To this end, some function approximation techniques make use of a *global approximation*, where the evaluation of the function at any point is based on some best fit to *all* of the training data. Whilst this ensures smooth functions that give a generally good approximation, they tend to smooth out local details, which can only be

inferred by considering the training data close to a given point, and ignoring training data that is far away. Thus, in order to capture these details a function with local approximation is desirable.

5.2.4 Rapid evaluation

Since space-searching techniques will be employed to scour parameter space for an appropriate solution, this will require many evaluations of g. Exactly how many is tied to the size of the population used in the search, which is in turn affected by the number of dimensions in $\mathcal{P}$, but it is reasonable to expect several hundred iterations of a population of, say, one thousand individuals — which would require several hundred thousand computations of g. To guarantee usability, a suitable function must therefore have complexity proportional to $O(|\mathcal{P}|)$, or at worst $O(|\mathcal{P}| \log |\mathcal{P}|)$ or similar.

5.2.5 Choice of function approximation technique

Having described what would be required of a suitable function approximation scheme, we now examine the schemes discussed in Chapter 4 and choose one which serves the needs of mapping between parameter space and adjective space. Table 4 highlights the techniques discussed in Chapter 4, and whether they satisfy the properties discussed thus far[3].

As can be seen from the table, only three techniques satisfy all the properties we are looking for. Of these, weighted least squares and RBFNs are generally better suited to user data, which will naturally be prone to error, as these techniques seek to find a better overall fit as opposed to interpolation techniques which will, by their nature, capture the inherent error.

In our technique, we use RBFNs rather than weighted least squares. RBFNs require less data in order to achieve a solvable set of linear constraints, and later in this chapter we show how an extension to the basic RBFN framework affords other benefits.

5.3 Representation of adjectives

As a model is being proposed that mathematically manipulates adjectival descriptions, the numerical representation of the descriptions is a key aspect of the approach. Chapter 3 posited the representation of an adjective as a single scalar value, which is derived from the notion that, when communicating, people use constructs such as the word “very” to quantify the extent to which an adjective applies — and numerically, this is akin to imposing a scale on the adjective.

However, an interface in which a user is required to associate a numerical value with an adjective may not be all that intuitive — the average person does not observe the world quantitatively, but

[3]Techniques that were deemed in Chapter 4 to be insuitable — for example due to poor scalability with higher dimensions — are excluded.

rather qualitatively. In fact, whilst people do on occasion use quantifiers such as "very" or "a little" to affect their descriptions, most often there is either a total lack of quantification, or the negative "not" is used to quantify the adjective.

During the initial course of this research, two investigations were conducted which in part addressed the issue of adjective representation. The full details of these can be found in Appendices A and B; the major point that they highlighted is that users would prefer a small, fixed number of states associated with each adjective. This can easily be addressed in one of two ways, either through *multicategory classification* [Kazmierczak and Steinbuch, 1963; Duda and Hart, 1973] or the *partitioning* of a single, scalar value's range into different states.

In the classical multicategory system, a *discriminant function* is assigned to each category, which takes a multidimensional feature vector and generates a single scalar output corresponding to the likelihood that the feature vectors map to the particular category. To decide which category a particular feature vector falls into, one simply chooses the category whose discriminant function has the greatest output.

Partitioning, on the other hand, retains the notion of a single scalar variable per adjective. The range of values taken on by this variable is split into a number of disjoint partitions, such that every point in the range lies in exactly one of the partitions. For example, in the extreme case of two states, the range would be split in half — with the lower half corresponding to the one state, and the upper half to the other (arbitrarily assigning the midpoint to one or the other partition). It should be clear, though, that as one progresses from one end of the range to the other through the various partitions, there should be a similar progression of logical interpretation of the partitions — it would not make sense, for example, to have the partitions corresponding to the concepts "good", "better" and "best", ordered according to their split of the range as "best", "good", "better".

In comparing and contrasting the two approaches, it should be noted that multicategory systems are more general, as they do not suffer from the restrictions imposed by partitioning. Multicategory systems overcome this by preserving the likelihood for *all* categories, which can be useful in providing information that indicates how strong a match has been found, or in cases where two or more categories have similarly strong values to indicate that the classification may be prone to error. Partitioning is, however, simpler, and for cases where there is a proportional correspondence between the logical states and their numerical representations, perhaps represents transitions between such states better than the multicategory approach.

Multicategory systems additionally require extra overhead, in the form of separate discriminant functions for every catgeory. This, coupled with the fact that only a very small number of categories are expected to be used in describing content, indicates that a partitioning system may be better suited for the purposes of adjective space.

Technique	Continuous	Generalises	Local	Rapid evaluation	Other comments
Shepard interpolant [Shepard, 1968]	yes	yes[1]	yes	yes	Guaranteed to interpolate the data points, possibly at the expense of a better overall fit.
Least squares [Levin, 1998]	yes	yes	no	yes	Requires a lot of data for higher dimensions and higher-degree polynomial fits
Weighted least squares [Levin, 1998]	yes	yes	yes[2]	yes[2]	Requires a lot of data for higher dimensions and higher-degree polynomial fits
Moving least squares [Levin, 1998]	yes	yes	yes	no	Every evaluation requires the solving of a linear system. Requires a lot of data for higher dimensions and higher-degree polynomial fits
RBFN [Orr, 1996]	yes	yes[3]	yes	yes	
Artificial neural networks [Werbos, 1974]	yes	unknown[4]	unknown[4]	yes	
LWPR [Vijayakumar et al., 2005]	yes	no[5]	yes	yes[5]	

Table 4: *A comparison of the various function approximation techniques discussed in Chapter 4, with reference to the desired function properties described in Section 5.2.*

[1] Although Shepard interpolants can generalise through the use of estimated gradient information, as one moves further from the convex hull the distances to the points in the training set converge, leading to the output simply being an average contribution from all the training points which may not be what one expects.

[2] A good local approximation for weighted least squares is dependent on there being a sufficient number of well placed weighting points. The number of weighting points, in turn, increases the complexity of evaluation, and so there is a trade-off between accuracy and speed of evaluation.

[3] RBFNs are capable of generalising, provided that the radial basis functions are placed so as to support the entire domain. Offsetting the approximation by first using a low-order polynomial approximation also aids in generalising.

[4] Due to the way in which ANNs incrementally adjust their weights, it is not possible to predict how well they will generalise or locally approximate a given set of data.

[5] The number of local models and hence, speed of evaluation, is a function of approximation parameters and the size of the training corpus — this is likely small, but unclear.

5.4 Choosing subsets of adjectives

Although it was mentioned in Chapter 3 that adjective space is likely of lesser dimensionality than parameter space, the number of adjectives available for the user to utilise could still be large — sufficiently large that to specify a value for every adjective in order to generate content may be extremely cumbersome. Users may, instead, simply wish for their content to reflect the description of only some small subset of adjectives. One way around this problem would be to impose a default values for each adjective, but this seems restrictive and on some level imposes a generic feel on the generated content. If the user deliberately did not choose specific adjectives, then it would be extremely useful if the system could simply ignore these in choosing an appropriate point in parameter space, from which the content is then generated.

Fortunately, the mapping of the inverse g affords this capability. In Section 5.1, it was shown that finding a point $\mathbf{p}$ in response to the user supplying point $\mathbf{a}$ amounted to minimising the equation

$$\begin{aligned} E(\mathbf{p}) &= \|g(\mathbf{p}) - \mathbf{a}\|^2 \\ &= \sum_i [g(\mathbf{p})_i - a_i]^2 \end{aligned}$$

That is, taking the sum-squared difference between $g(\mathbf{p})$ and $\mathbf{a}$ over all the dimensions of $\mathcal{A}$. A simple modification gives the new error metric

$$E_A(\mathbf{p}) = \sum_i m_i [g(\mathbf{p})_i - a_i]^2 \tag{10}$$

where $m_i = 1$ if the adjective corresponding to the i^{th} dimension of $\mathcal{A}$ was chosen by the user, and $m_i = 0$ if it was not chosen. That is, when computing the error of the function, only the adjectives in which the user is interested are considered — the other adjectives are free to take on whatever values result in the minimisation of the error metric.

5.5 Dynamic use of additional adjectives

The presentation of adjective space thus far has made use of a fixed set of adjectival descriptors which the user is forced to use. Whilst this is motivated by the need to reduce the complexity of the problem and a desire for objectivity in response to users' observations, it does confer an element of bias by suggesting to the user what descriptors they should be using. It may additionally be seen as being overly restrictive, by not allowing for other possibly valid descriptors that may seem more

natural to the user. Whilst one could address this problem by enlarging adjective space to include all possible descriptors, this is a restrictive solution and does not support the evolution of language to adopt new adjectives.

The major difficulty in supporting new descriptors is in establishing relations to other known descriptors — if one were able to do this, then some degree of information could be inferred about the new descriptor so as to facilitate its usage. By restricting the set of descriptors to adjectives in the English language, it is in fact possible to obtain these relationships through the use of a system such as WordNet [Fellbaum, 1998], introduced in Chapter 3.

WordNet is, in essence, a database of words in the English language, complete with parts of speech, meanings, example sentences and relational information. It groups words into what are referred to as *synsets* (synonym sets), based not on the spelling of words but the *sense* in which they are used. For example, the word "cold" can be used in a number of contexts or senses, such as to describe temperature or alternatively to describe emotion, and thus is a member of multiple synsets. Each synset has optional links to other synsets to reflect a variety of relationships, such as antonyms, similar meanings, derived forms or compound words.

As such, WordNet is a powerful tool that provides a means for new adjectival descriptors to be related to existing ones, through the traversal of the relationship graph inherent in the database. What remains is mathematically defining how new descriptors can seamlessly fit into the framework presented thus far — this is addressed in the following section, in which an extension to RBFNs is proposed that supports the use of WordNet data, as well as providing additional benefits.

5.6 Augmenting training data with certainty values

The issue of differing perceptions, discussed in Chapter 3, is an important problem that should be addressed. One possible solution is now proposed, after which it is shown how this solves not only the problem of differing perceptions but also allows for seamless integration of WordNet data as discussed in the previous section.

The key idea is to extend the set of training data to include a *certainty value*, k_i, that is associated with each pair $(\mathbf{p_i}, z_i)$. As the name implies, k_i measures the certainty of the observation, and is utilised mathematically to weight the least square error of g

$$C = \sum_i k_i \|g(\mathbf{p_i}) - z_i\|^2$$

Certainty values thus allow the function approximation to more closely approximate regions of certain data, with the flexibility to diverge in regions of less certain data if this provides a better fit.

5.6.1 Applying certainty values to differing perceptions

In Chapter 3, two methods were proposed for dealing with the problem of differing perceptions that involved the use of data from multiple sources — by either collecting data from many people and thus achieving a synthesis of opinion, or by taking a pre-determined function that reflects the artist's perceptions and then, through a small set of additional training data, tailoring the function to match a specific user's expectations.

In collecting data from many users, it often suffices to give each training point an equal weight. Statistical analysis might, however, reveal outliers in the data that could have a negative influence on the function's overall fit — yet at the same time these outliers should be considered as important parts of the data. Certainty values are well suited here as outliers can be tagged as having lower certainty, thus affecting the resulting function less. It may also be desirable to assign the data of particular expert users greater importance — to this end, certainty values can again be employed by tagging such data with larger values, causing the function approximation to more closely approximate these points.

Certainty values are possibly most intuitively utilised, however, in tailoring an existing function to better match an individual user's unique perceptions. This is achieved by presenting the user with a small set of generated content to which they assign adjectival descriptors, providing additional training data that specifically captures this user's perceptions. To coerce the function approximation into adapting to this specific user, the additional training data is simply assigned a higher certainty value than those of the main training corpus. In this way, the resulting function still makes use of the data provided by the artist, but places more emphasis on the user-specified data.

5.6.2 Applying certainty values to WordNet

Armed with the extension of certainty values, the semantic relationships provided by WordNet can be harnessed in two distinct ways:

1. **Amplification of adjective space.** New adjectival descriptors can be added as a pre-process to the set $\mathbf{A}$ of available descriptors, by following links in the semantic relationship graph. More specifically, suppose that the descriptor $\alpha \in \mathbf{A}$ were used to describe a particular piece of content, with certainty k_A. We could then posit that the content could also be described by adjectives that are similar to α, only with a lesser certainty dk_A, where $0 < d < 1$. This can be applied to antonyms, too — if α were associated with scalar value a, and β was an antonym of α, then we could suggest that the content also be described by β associated with scalar value $-a$ and certainty value dk_a.

2. **Interpretation of new adjectival descriptors.** Using certainty values, users can specify new adjectives that are in the WordNet database but not in $\mathbf{A}$. Suppose the user specified an adjective β: by traversing the semantic network provided by WordNet, the minimum distance

l from β to an element $\alpha \in \mathbf{A}$ can be found. Any content that is tagged with the adjective α, can then also be tagged with β — but with certainty scaled by d^l.

5.6.3 Incorporating certainty values into radial basis function networks

We now discuss how certainty values can be incorporated into a RBFN. We will refer to RBFNs that have been augmented with certainty values as *certainty radial basis function networks*, or CRBFNs.

Solving for the weight vector w

Recall that a regularised RBFN seeks to minimise the error function

$$C = \sum_{i=1}^{n} \|g(\mathbf{p_i}) - z_i\|^2 + \sum_{j=1}^{m} \lambda_j w_j^2 \tag{11}$$

When incorporating certainty values, this is modified slightly to instead give

$$C' = \sum_{i=1}^{n} k_i \|g(\mathbf{p_i}) - z_i\|^2 + \sum_{j=1}^{m} \lambda_j w_j^2 \tag{12}$$

The derivation of the weights w_j, that correspond to the minimum value of C', is quite similar to the solution for the case without certainty values (covered in detail by Orr [1996]). In order to minimise C' and solve for the w_j, one has to differentiate with respect to each w_j and equate to 0, giving

$$\frac{\partial C}{\partial w_j} = 2\sum_{i=1}^{n} \left[k_i \left[g(\mathbf{p_i}) - z_i \right] \frac{\partial g}{\partial w_j}(\mathbf{p_i}) \right] + 2\lambda_j w_j \tag{13}$$

Since $g(\mathbf{p_i}) = \sum_{j=1}^{m} w_j h_j(\mathbf{p_i})$, $\frac{\partial g}{\partial w_j} = h_j(\mathbf{p_i})$. Substituting into Equation 13 and equating to 0 gives

$$\sum_{i=1}^{n} \left[k_i g(\mathbf{p_i}) h_j(\mathbf{p_i}) \right] + \lambda_j w_j = \sum_{i=1}^{n} k_i z_i h_j(\mathbf{p_i})$$

By using vector notation, this equation can be more compactly expressed as

$$\mathbf{h_j}^\top \mathbf{K g} + \lambda_j w_j = \mathbf{h_j}^\top \mathbf{K z}$$

where

$$\mathbf{h_j} = \begin{bmatrix} h_j(\mathbf{p_1}) \\ h_j(\mathbf{p_2}) \\ \vdots \\ h_j(\mathbf{p_n}) \end{bmatrix}, \quad \mathbf{K} = \begin{bmatrix} k_1 & 0 & \cdots & 0 \\ 0 & k_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & k_n \end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix} g(\mathbf{p_1}) \\ g(\mathbf{p_2}) \\ \vdots \\ g(\mathbf{p_n}) \end{bmatrix}, \quad \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}$$

Since there are m of these equations (one for each weight w_j), they can be combined into a single equation

$$\begin{bmatrix} \mathbf{h_1}^\top \mathbf{K g} \\ \mathbf{h_2}^\top \mathbf{K g} \\ \vdots \\ \mathbf{h_n}^\top \mathbf{K g} \end{bmatrix} + \begin{bmatrix} \lambda_1 w_1 \\ \lambda_2 w_2 \\ \vdots \\ \lambda_n w_n \end{bmatrix} = \begin{bmatrix} \mathbf{h_1}^\top \mathbf{K z} \\ \mathbf{h_2}^\top \mathbf{K z} \\ \vdots \\ \mathbf{h_n}^\top \mathbf{K z} \end{bmatrix}$$

which can further be simplified to the matrix equation

$$\mathbf{H}^\top \mathbf{K g} + \mathbf{\Lambda w} \quad = \quad \mathbf{H}^\top \mathbf{K z} \tag{14}$$

where $\mathbf{H}$ and $\mathbf{\Lambda}$ are the matrices given in Chapter 4 and defined as

$$\mathbf{H} = \begin{pmatrix} h_1(\mathbf{p_1}) & \cdots & h_m(\mathbf{p_1}) \\ \vdots & \ddots & \vdots \\ h_1(\mathbf{p_n}) & \cdots & h_m(\mathbf{p_n}) \end{pmatrix}, \quad \mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{pmatrix}$$

Now, since $g_i = g(\mathbf{p_i}) = \sum_{j=1}^{m} w_j h_j(\mathbf{p_i})$, g_i can be expressed as $g_i = \bar{\mathbf{h_i}}^\top \mathbf{w}$ where

$$\bar{\mathbf{h_i}} = \begin{bmatrix} h_1(\mathbf{p_i}) \\ h_2(\mathbf{p_i}) \\ \vdots \\ h_m(\mathbf{p_i}) \end{bmatrix}$$

But the $\bar{\mathbf{h_i}}$ are simply the rows of $\mathbf{H}$, and so

$$\mathbf{g} = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{h}}_{\mathbf{1}}^{\top}\mathbf{w} \\ \bar{\mathbf{h}}_{\mathbf{2}}^{\top}\mathbf{w} \\ \vdots \\ \bar{\mathbf{h}}_{\mathbf{p}}^{\top}\mathbf{w} \end{bmatrix} = \mathbf{H}\mathbf{w}$$

Substituting this result into Equation 14 gives

$$\begin{aligned} \mathbf{H}^{\top}\mathbf{K}\mathbf{z} &= \mathbf{H}^{\top}\mathbf{K}\mathbf{H}\mathbf{w} + \mathbf{\Lambda}\mathbf{w} \\ &= \left[\mathbf{H}^{\top}\mathbf{K}\mathbf{H} + \mathbf{\Lambda}\right]\mathbf{w} \end{aligned}$$

which has the solution

$$\begin{aligned} \mathbf{w} &= \left[\mathbf{H}^{\top}\mathbf{K}\mathbf{H} + \mathbf{\Lambda}\right]^{-1}\mathbf{H}^{\top}\mathbf{K}\mathbf{z} \\ &= \mathbf{A}^{-1}\mathbf{H}^{\top}\mathbf{K}\mathbf{z} \end{aligned}$$

where $\mathbf{A} = \mathbf{H}^{\top}\mathbf{K}\mathbf{H} + \mathbf{\Lambda}$.

Determining the projection matrix P

As demonstrated in Chapter 4, the projection matrix is used extensively in various aspects of a complete RBFN solution, ranging from the optimisation of regularisation parameters, to centre selection. It is thus imperative to obtain a closed form for the projection matrix that takes into account certainty values. Fortunately, the derivation is straightforward and, as with the solution for the weight vector $\mathbf{w}$, is a simple extension to the derivation for a regular RBFN.

In a regular RBFN, the projection matrix satisfies the two equations

$$\begin{aligned} C &= \mathbf{z}^{\top}\mathbf{P}\mathbf{z} \\ \hat{S} &= \mathbf{z}^{\top}\mathbf{P}^2\mathbf{z} \end{aligned}$$

where C is the cost function of Equation 11, and $\hat{S}$ is the sum-squared error of the model also given

by

$$\hat{S} = \sum_{i=1}^{n} \|g(\mathbf{p_i}) - z_i\|^2$$

The sum-squared error is of particular importance, as this is used by the various model selection criteria to optimise the network.

By including certainty values, the sum-squared error still maintains its compact representation in terms of the projection matrix, whilst the cost function becomes only slightly more complex. Recall that in vector notation, $\hat{S}$ can be expressed as

$$\hat{S} = (\mathbf{z} - \mathbf{g})^\top (\mathbf{z} - \mathbf{g})$$

Now from the derivation of the optimal weight vector $\mathbf{w}$, we have

$$\begin{aligned} \mathbf{g} &= \begin{bmatrix} \bar{\mathbf{h_1}}^\top \mathbf{w} \\ \bar{\mathbf{h_2}}^\top \mathbf{w} \\ \vdots \\ \bar{\mathbf{h_p}}^\top \mathbf{w} \end{bmatrix} \\ &= \mathbf{Hw} \\ &= \mathbf{HA^{-1}H^\top Kz} \end{aligned}$$

and therefore

$$\begin{aligned} \mathbf{z - g} &= \mathbf{z - HA^{-1}H^\top Kz} \\ &= \left[\mathbf{I_n - HA^{-1}H^\top K}\right] \mathbf{z} \\ &= \mathbf{Pz} \end{aligned}$$

where the projection matrix $\mathbf{P}$ is given by

$$\mathbf{P} = \mathbf{I_n - HA^{-1}H^\top K}$$

Hence,

$$\begin{aligned}\hat{S} &= (\mathbf{z}-\mathbf{g})^\top(\mathbf{z}-\mathbf{g}) \\ &= \mathbf{z}^\top\mathbf{P}^\top\mathbf{P}\mathbf{z}\end{aligned}$$

Now, remembering that $\mathbf{A} = \mathbf{H}^\top\mathbf{K}\mathbf{H} + \mathbf{\Lambda}$ and the fact that $\mathbf{K}$ and $\mathbf{\Lambda}$ are diagonal and hence symmetric, it follows that $\mathbf{A}$ must also be symmetric. $\mathbf{P}$ is thus similarly symmetric ($\mathbf{P}^\top = \mathbf{P}$) and therefore

$$\begin{aligned}\hat{S} &= \mathbf{z}^\top\mathbf{P}^\top\mathbf{P}\mathbf{z} \\ &= \mathbf{z}^\top\mathbf{P}^2\mathbf{z}\end{aligned}$$

as with a regular RBFN.

In vector notation, the cost function is equivalent to

$$\begin{aligned}C &= (\mathbf{z}-\mathbf{g})^\top(\mathbf{z}-\mathbf{g}) + \mathbf{w}^\top\mathbf{\Lambda}\mathbf{w} \\ &= \hat{S} + (\mathbf{z}^\top\mathbf{K}\mathbf{H}\mathbf{A}^{-1})(\mathbf{A}-\mathbf{H}^\top\mathbf{K}\mathbf{H})(\mathbf{A}^{-1}\mathbf{H}^\top\mathbf{K}\mathbf{z}) \\ &= \hat{S} + \mathbf{z}^\top\mathbf{K}\left[(\mathbf{H}\mathbf{A}^{-1}\mathbf{H}^\top\mathbf{K}) - (\mathbf{H}\mathbf{A}^{-1}\mathbf{H}^\top\mathbf{K})(\mathbf{H}\mathbf{A}^{-1}\mathbf{H}^\top\mathbf{K})\right]\mathbf{z} \\ &= \mathbf{z}^\top\mathbf{P}^2\mathbf{z} + \mathbf{z}\mathbf{K}(\mathbf{P}-\mathbf{P}^2)\mathbf{z} \\ &= \mathbf{z}\left[(\mathbf{I_n}-\mathbf{K})\mathbf{P}^2 + \mathbf{K}\mathbf{P}\right]\mathbf{z}\end{aligned}$$

It can be easily seen from this equation that when all the certainty values are 1 (representing a regular RBFN) then the cost function has the form $\mathbf{z}^\top\mathbf{P}\mathbf{z}$ as before.

5.7 Complete overview of technique

Much of the overall technique presented in this thesis has already been alluded to, with reference to addressing particular concerns at various stages. The system as a whole is now presented and discussed.

In short, the use of adjective space to generate procedural content comprises the following aspects:

1. **Adjective representation.** Adjective space is conceptually a higher-dimensional space in

which each dimension is tied to a specific synset drawn from the WordNet database. The act of describing a piece of content is achieved by choosing adjectives which apply to that content, and optionally quantifying the extent to which they apply through a scalar value in the range $[0; 1]$. Negative quantifications in the range $[-1; 0)$ are also allowed, which indicate an absence of that adjective in the content. Dependent on which of the three ways the descriptions in adjective space are elicited (per-user training, synthesis of opinion or artist's vision), each description also has a certainty value attached to it. Certainty values typically lie in the range $[0; 1]$, but in theory are unconstrained.

2. **Training.** A number of points are selected from parameter space, $\mathcal{P}$, and the digital content associated with these points is generated. The content is then tied to points in adjective space, $\mathcal{A}$, in one of three possible ways:

   (a) **Per-user training.** Each user is required to associate every piece of content with a description in adjective space.

   (b) **Synthesis of opinion.** Many users are presented with a small subset of the content, and associate each of these with an appropriate description in adjective space.

   (c) **Artist's vision.** A single artist/designer or small group of people review every piece of content, and associate each with an appropriate description in adjective space.

   Having completed this data collection process, the system is now in a position to be trained as a pre-process. For each dimension in adjective space, we have a set of points $\mathcal{T} = \{\mathbf{t_1}, \mathbf{t_2}, \ldots, \mathbf{t_n}\}$ corresponding to outputs $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ with associated certainty values $\mathbf{K} = \{k_1, k_2, \ldots, k_n\}$. This data is used to train a number of CRBFNs, one for each adjective given in the descriptions. This completes the preprocess required before adjective space can be used to generate content.

3. **Adjective selection.** The user selects one or more adjectives describing the content that they wish to generate. They optionally quantify the adjectives using a scalar value or category associated with each adjective.

4. **Particle swarm optimisation.** Given the description supplied by the user, the goal is now to find a point in parameter space that maps as closely to thie description as possible, as defined by the error metric in Equation 10. This is achieved using particle swarm optimisation to find a point $\mathbf{p}$ that minimises the metric.

5. **Content generation.** Finally, the point $\mathbf{p}$ is fed into the procedural generation system, producing content. Having created content, the user may at this stage repeat steps 3-5, refining the content generated until it is satisfactory. Since the adjectival interface serves as a form of intrinsic scaffolding, it can at any stage be stripped away to give the user access to the underlying procedural parameters.

5.8 Summary

In this chapter, we have addressed the finer details of using function approximation for the purpose of mapping between adjective space and parameter space, and overcome the problems set out in earlier chapters. This was in part achieved through the use of CRBFNs, which are a novel extension of RBFNs to support the association of certainty values with the training data. What remains is to test the efficacy of our technique, which we explore in the following chapter.

Chapter 6

Experimental testing

To evaluate the adjectival interface presented in this thesis, several investigations were conducted. The first two investigations were used to elicit user feedback and guide the research, and as such do not provide an evaluation of the final technique. For completeness, the details of these investigations are provided in Appendices A and B; this chapter is devoted to the final experiment used to validate the technique.

To determine whether the adjectival interface is of actual benefit to users, it is compared and contrasted to an interface which offers direct control over procedural parameters. This is accomplished by addressing two questions:

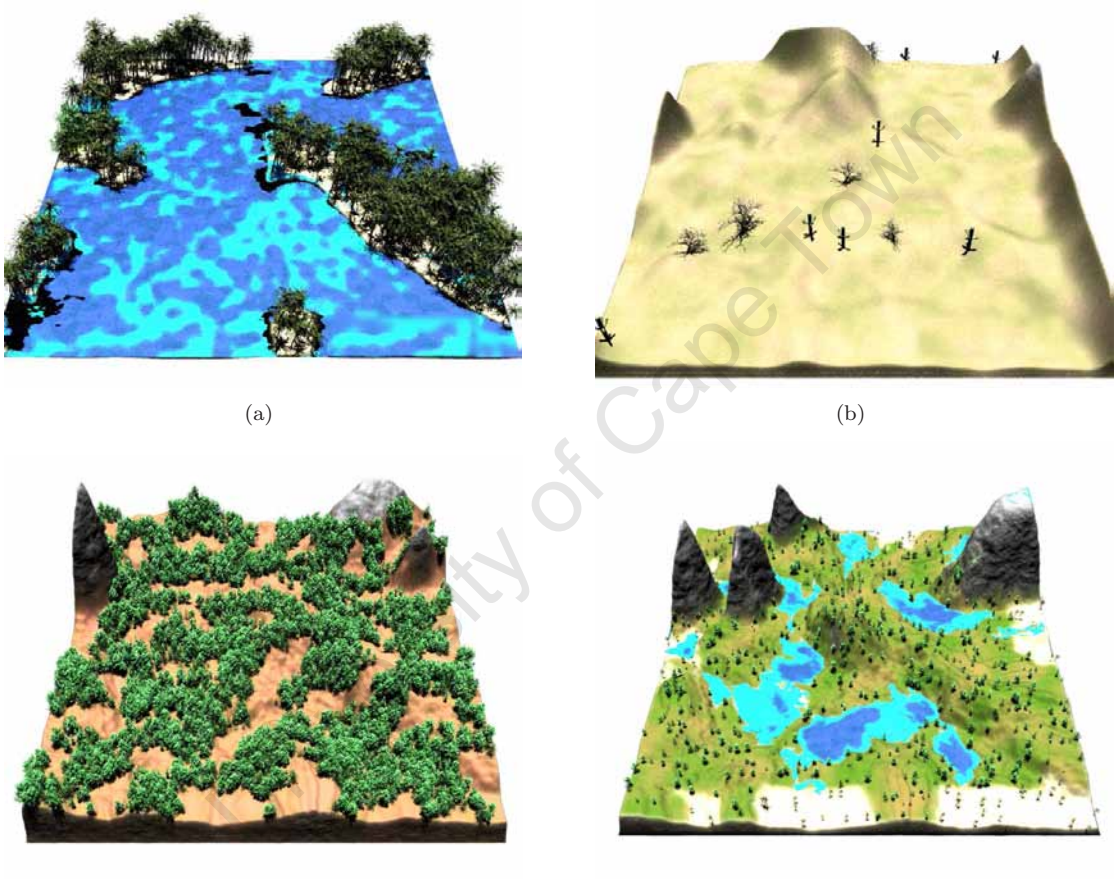
1. **How effective is the interface for the rapid generation of procedural content?** Given that an interface for content generation is being presented, it is natural to determine how satisfied users are with the content produced. Did the users get frustrated with the interface? Did the interface meet their expectations? Were they satisfied? These are just some questions that might be asked; the exact questions and hypotheses tested are detailed in due course.

2. **Does the adjectival interface more faithfully produce what the user wishes to create, compared with the direct specification of procedural parameters?** Whilst the interface itself might be deemed superior to specifying values for individual parameters, it is possible that this enhancement comes at the sacrifice of quality in output. It is thus important to also compare the two interfaces in terms of the procedural content produced, and ascertain some means for doing so objectively.

With these thoughts in mind, a detailed experimental setup is now described that addresses the questions posited above.

6.1 Experimental design

Since the major focus in this thesis has been on procedural graphics, a procedural system that creates outdoor landscapes was chosen for the experiment. This system was created using the software package Houdini [Side Effects Software Inc., 2008b], chosen for the procedural nature in which it allows 3D graphics content to be created. In total, 49 procedural parameters were exposed, which allowed for the creation of a diverse range of landscapes: Figure 46 shows some examples.



(a) (b)

(c) (d)

Figure 46: *Examples of the variety of landscapes that can be generated with the procedural system used in the experiment.*

Key components of the overall system included the following:

1. **Terrain.** Through Houdini's compositing engine, several different parts of the terrain — such as escarpment, mountains and low-amplitude noise — could be individually crafted and combined to give a very complex landscape. Gradient and height information were used to assign various portions of the landscape to different vegetation zones with appropriate texturing, and the final landscape was rendered using variable subdivision dependent on the vegetation zone. This allowed, for example, the preservation of creasing in rocky areas.

2. **Rivers, lakes and sea.** Using the final terrain, basins where water would collect could be identified, as could rivers that would drain the water and eventually lead to the sea. This is accomplished using a custom-built plugin for Houdini that simulates the rise of the sea-level, and thus determines where water pools by observing which parts of land become flooded as the sea-level rises.

3. **Trees and vegetation.** Once the terrain has been generated, and rivers and lakes have been placed, the system is in a position to place a number of shrubs and trees in appropriate areas. Houdini's compositing engine is used to identify different types of landscape area — such as beach, temperate, desert and mountainous — which, coupled with parameters that control the density of vegetation in these regions, is used to randomly scatter vegetation in appropriate locations.

Further details of the procedural system, as well as a full list of the parameters, can be found in Appendix C.

Having created a procedural modelling system for generating virtual landscapes, 500 points in the parameter space of the model were randomly chosen and their respective landscapes generated. These were then described using a set of 22 adjectival descriptors, providing training data for the CRBFN optimisation. Using WordNet to extrapolate to semantically connected synsets with a decay factor of $d = 0.7$, a total of 81 adjectival descriptors were made available via an adjectival interface. A two-stage experimental design was then developed in order to address the questions posed, using this system.

6.1.1 First stage

Each user was assigned one interface with which to produce content. This was either the adjectival interface or an interface allowing for the direct specification of the procedural parameters. In this instance, direct specification does not refer to the use of the native Houdini interface: instead, the users are presented with several slider widgets giving them control over the individual real-valued numeric parameters.

After being given a brief introduction in which the user was presented with the appropriate interface and an example of its usage, they were shown a photograph of a real-life outdoor landscape. Their goal was to use the interface provided to create a virtual landscape that depicted the landscape shown in the photograph as faithfully as possible.

As one goal for the technique is to allow for rapid content generation, users were limited to 22 minutes of working time per photograph, during which they could procedurally generate several virtual landscapes and ultimately pick the one which best captured the photograph. This process was repeated with each user for a second, different photograph, again limiting the user to 22 minutes of working time. Once the user had completed working on their second task, they were asked a

number of questions related to their experience, listed in Table 5. The choice of 22 minutes was so as to maximise the time that users could spend on each task, where each user was given an hour long period for their overall participation. This allowed for 10 minutes to cover the initial introduction and for the user to answer the final questionnaire, and gave a few minutes extra to deal with any unforeseen problems.

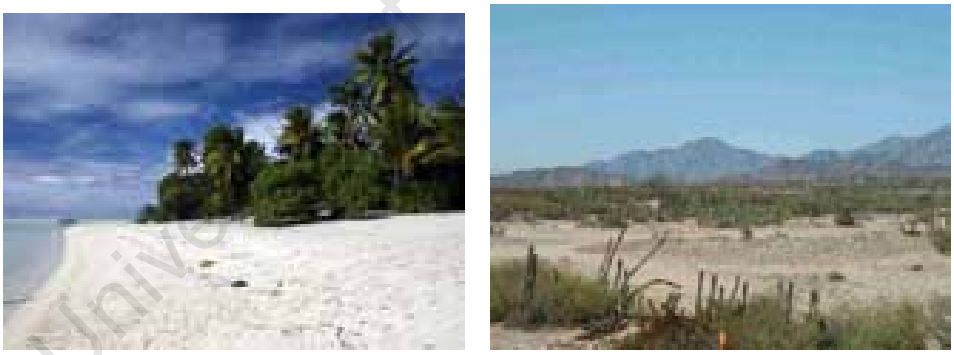
1. On a scale of 1 to 10 (with 10 being better), how faithfully do you feel the first virtual landscape that you created matched the content of the first photograph?
2. On a scale of 1 to 10 (with 10 being better), how faithfully do you feel the second virtual landscape that you created matched the content of the second photograph?
3. On a scale of 1 to 10, how easy to understand did you find the interface? (1 is hard, 10 is easy)
4. On a scale of 1 to 10, how easy was the interface to use? (1 is hard, 10 is easy)
5. On a scale of 1 to 10, how frustrated did you get while using the system? (1 is not frustrated, 10 is very frustrated)
6. On a scale of 1 to 10, how often did the virtual landscapes generated meet your expectations? (1 is never, 10 is always)
7. Do you feel that you needed more time than you were given on each photograph to be able to create a satisfactory virtual landscape?
   (a) If you answered yes to the previous question, how much extra time in minutes do you think you would have needed?
8. Do you think that with practise using this system you would be able to decrease the time it takes you to generate a virtual landscape?
   (a) If you answered yes to the previous question, how long in minutes do you think it would take you to generate a virtual landscape that matches a new photograph (after lots of practise in using the system)?

Table 5: *The list of questions asked of users in the first stage of the experiment.*

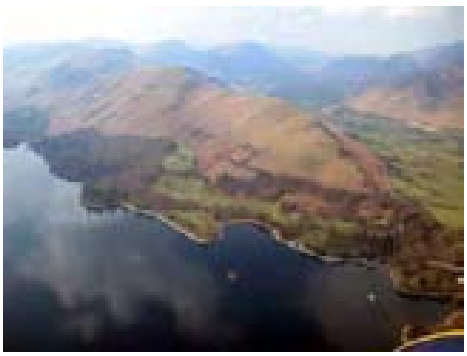
Five photographs of real-life landscapes were chosen to reflect a diversity in landscapes, as well as landscapes that could plausibly be captured by the procedural system. The photographs used are shown in Figure 47. These were evenly distributed in a random assignment to the users that took part in the experiment.

The interface presented to users who were assigned the direct specification interface can be seen in Figure 48. Users of the adjectival interface could add adjectives from a list as shown in Figure 49, and then adjust aspects of each adjective such as its quantification and negation as shown in Figure 50.

(a) (b)



(c) (d)



(e)

Figure 47: *Photographs of real-life landscapes presented to users for the experiment.*

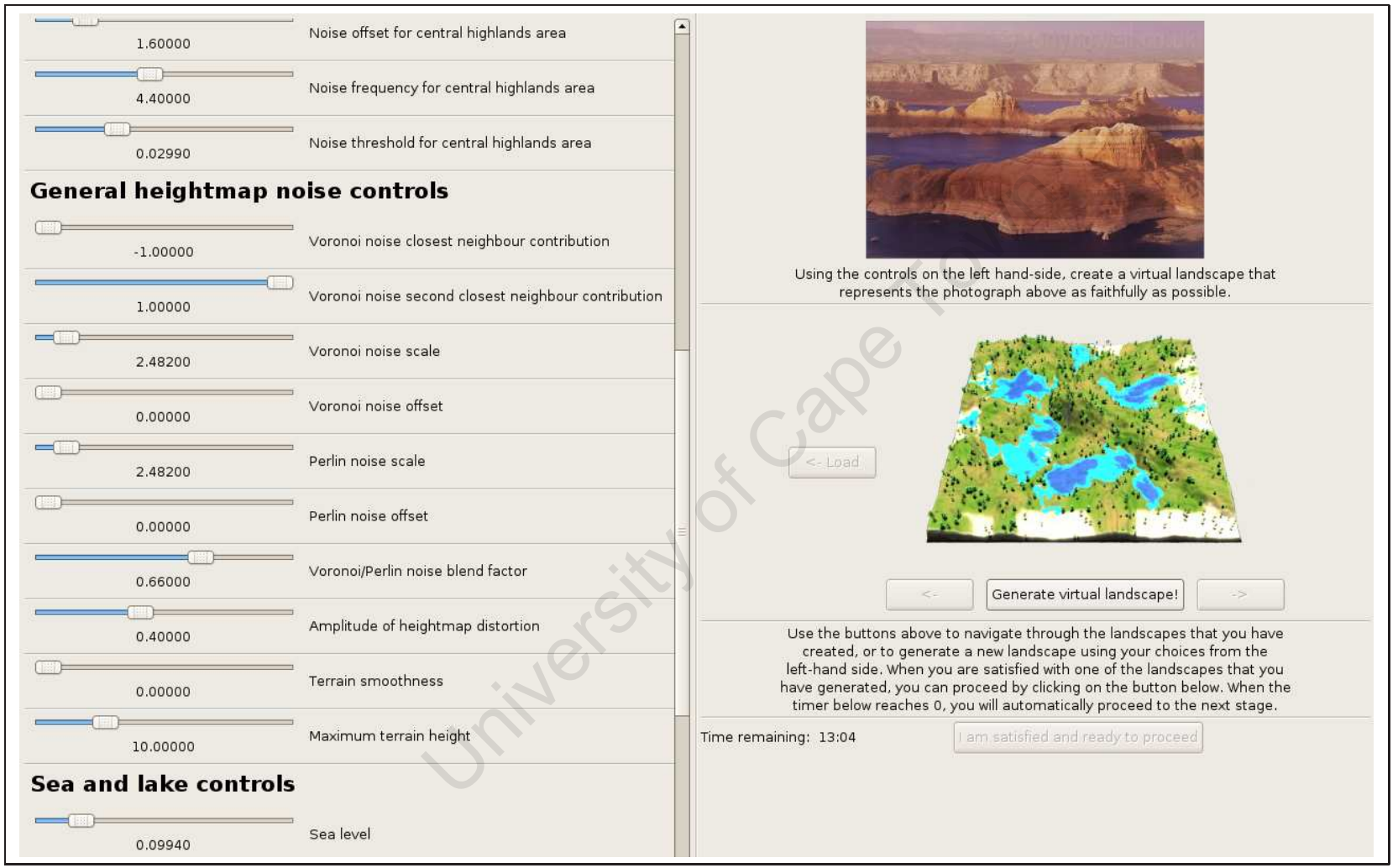


Figure 48: *Interface allowing for the direct specification of procedural parameters.*

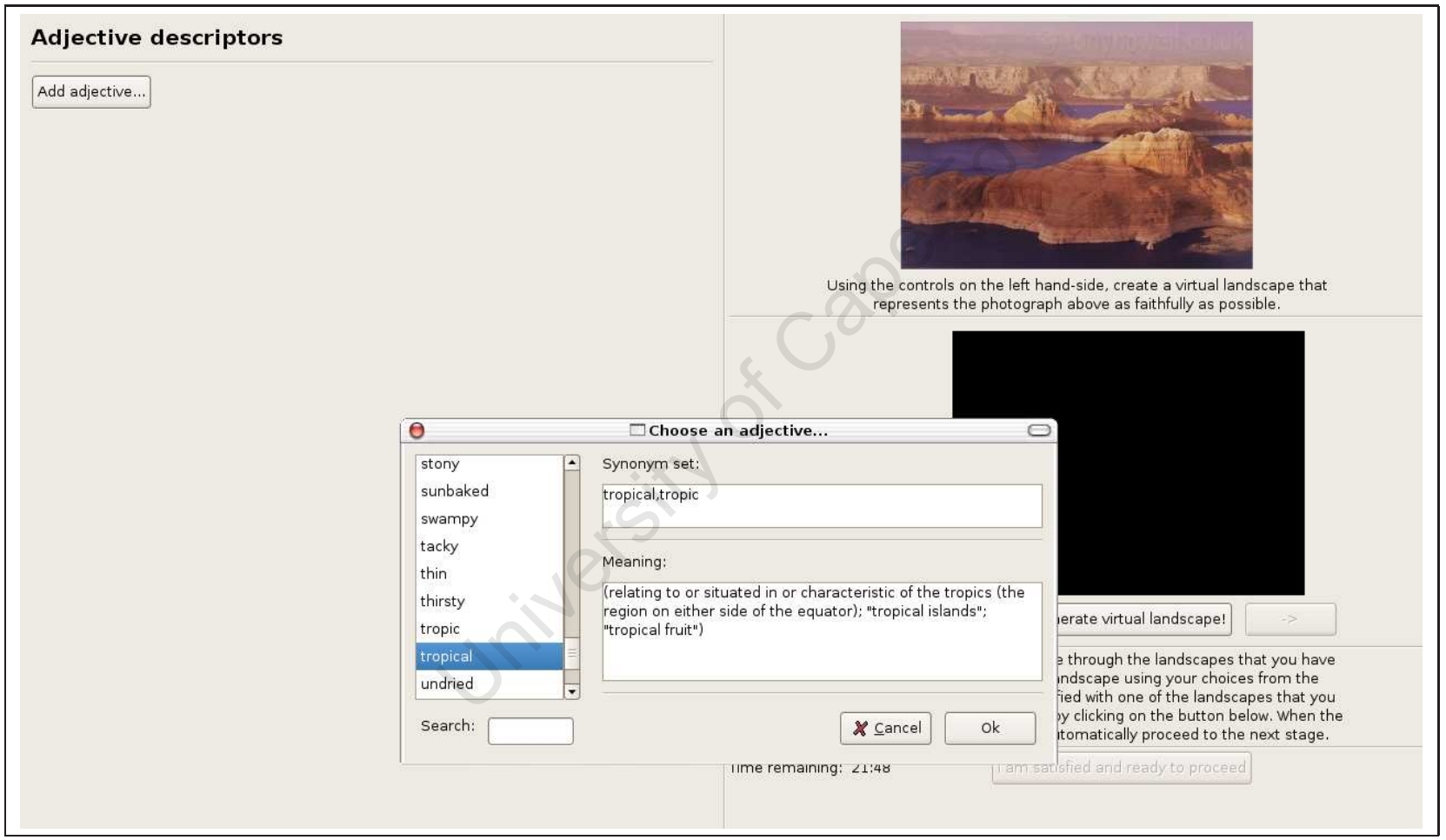


Figure 49: *Adjectival interface for choosing an adjective.*

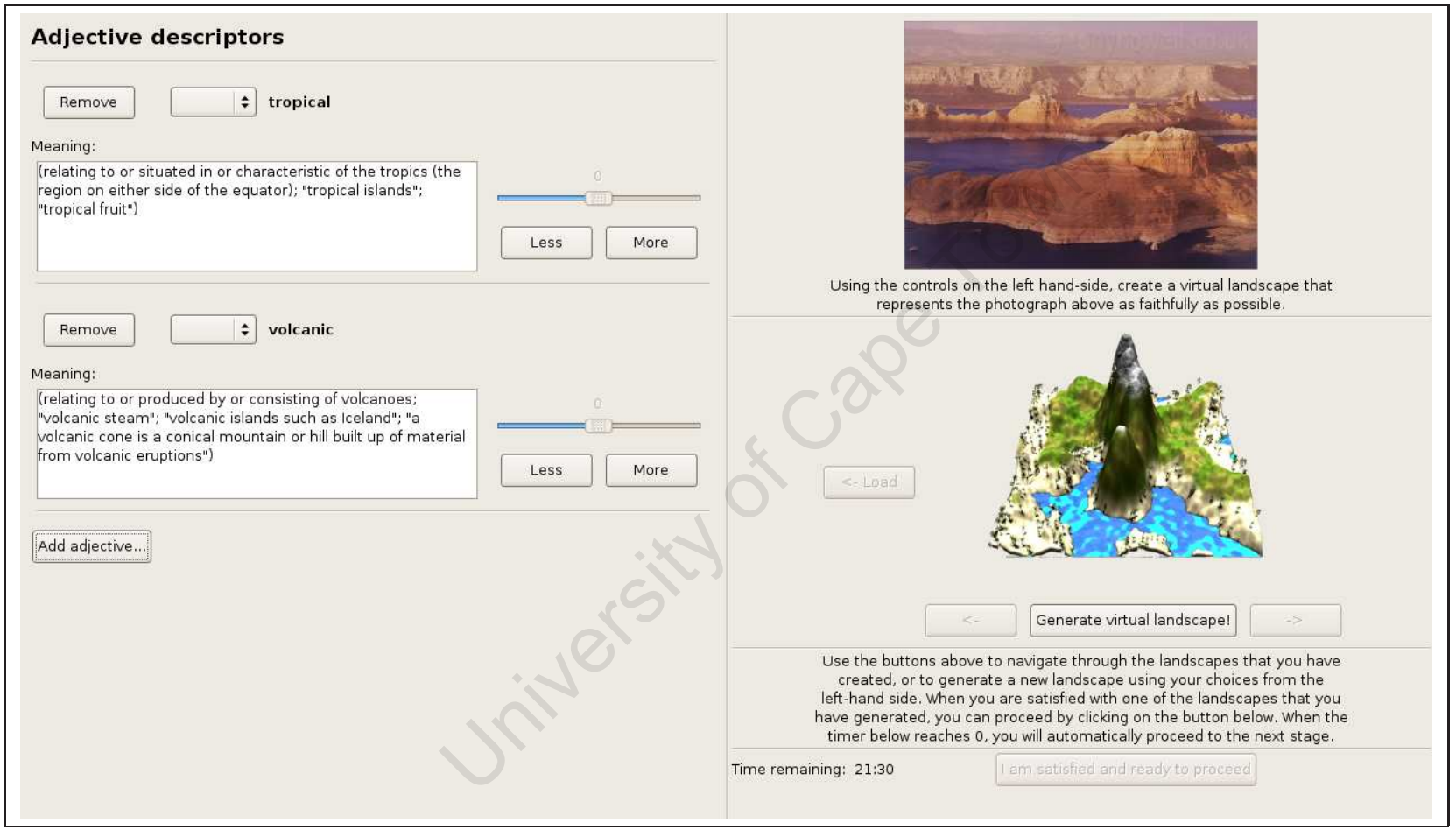


Figure 50: *Adjectival interface for adjusting the quantification and negation of adjectives.*

6.1.2 Second stage

To objectively evaluate which of the two interfaces allows users to create virtual landscapes that more faithfully capture the photographs shown, an independent study was conducted in the second stage of the experiment. Participants in the study were shown the photographs used for the first stage, and with each photograph they were shown two virtual landscapes generated by the users in the first stage of the experiment — one created using the adjectival interface and one created using the direct parameter specification interface. The task for participants in the second stage was simply to pick the virtual landscape that most faithfully captured the photograph shown. The virtual landscapes displayed were randomly selected from the sets available, and the order in which the two virtual landscapes were shown was also randomly assigned. As such, there was no way for participants to know which landscape was generated using each interface — and in fact, they were not even told how the images were generated, and so had no idea that the images had even been created by human users.

6.2 Objectives

By constructing the experiment in these two stages, the questions posed at the beginning of this chapter are successfully addressed. The first stage serves as both a qualitative and quantitative comparator of the two interfaces — qualitative by virtue of the answers provided by users in response to the questionnaire, but also quantitative through the implicit analysis of the time taken by the users to complete their tasks. In short, the first stage provides an effective means to judge which interface is easier for users to utilise.

Of course, an intuitive interface that produces poor procedural content would be totally ineffective, and so further analysis to compare the output generated is necessary. That is the purpose of the second stage, which was carefully designed so as to ensure an objective comparison. By making use of a second, independent set of participants, the second stage avoided any possible issues with learning bias that may have resulted from users in the first stage being called on for a comparitive assessment. The independent user base also ensured that users were not giving preference to content which they had generated themselves. Furthermore, by simply presenting participants with a photograph and two images of generated content, no detail about the differences between the images was exposed, which might otherwise have subjectively affected the participants reactions.

6.3 Hypotheses

In order to properly evaluate the statistical significance of the results obtained through this experiment, it is important at the outset to carefully choose hypotheses that are to be tested.

The expectation is that the adjectival interface should provide a more intuitive and user-friendly experience. One can therefore hypothesise that for the subjective measures of the first stage, the adjectival interface should outperform the direct specification interface, and therefore that:

1. **Users will find the adjectival interface easier to understand.**
2. **Users will find the adjectival interface easier to use.**
3. **Users will find the adjectival interface less frustrating to use.**
4. **Users will find that the adjectival interface generates landscapes that more closely match the photograph.**
5. **Users of the adjectival interface will feel more strongly that the landscapes generated met their expectations.**
6. **Users of the adjectival interface will feel that they require less time to perform the task than users of the direct specification interface.**

With sufficient training, users might learn how to use the direct specification interface sufficiently well, and so it is not as easy to judge which user group will perform better. We can, however, hypothesise that there is likely to be some difference:

7. **After sufficient training, the average times that it would take users of each interface to complete the task, are not equal.**

Although users in the first stage are expected to have a natural tendency to prefer the adjectival interface, this does not suggest that the resulting output is necessarily more faithful to the task than that generated by the other user group. One cannot, therefore, presuppose which of the two interfaces will lead to better content — but we can hypothesise that the choices made by participants were not entirely random. That is:

8. **The probability of preferring a landscape generated using the adjectival interface, over a landscape generated using the direct specification interface, is not equal to** 0.5**.**

6.4 Results and discussion

6.4.1 First stage

In total, 35 subjects took part in the first stage of the experiment — 17 were assigned the direct specification interface, and 18 were assigned the adjectival interface. The responses to the questionnaire for the direct specification group are listed in Tables 6 and 8, and those for the adjectival

	Accuracy 1	Accuracy 2	Easy to understand	Easy to use	Frustration	Expectations met	Combined score
	6	4	7	7	6	4	22
	8	5	4	10	8	6	25
	1	5	3	7	3	5	18
	5	3	2	3	7	4	10
	1	3	1	2	6	3	4
	6	4	7	7	1	5	28
	5	5	2	3	6	3	12
	6	7	6	9	4	4	28
	6	5	5	6	2	3	23
	3	6	3	8	8	2	14
	2	4	2	3	8	2	5
	3	2	3	10	3	1	16
	3	5	3	3	5	5	14
	4	5	6	8	6	5	22
	1	5	1	2	7	3	5
	6	5	4	7	6	5	21
	6	8	6	10	4	6	32
Mean	4.24	4.76	3.82	6.18	5.29	3.88	17.59
Std deviation	2.17	1.44	2.01	2.92	2.14	1.45	8.58

Table 6: *Responses from users of the direct specification interface in the first stage for questions 1-6.*

	Accuracy 1	**Accuracy 2**	**Easy to understand**	**Easy to use**	**Frustration**	**Expectations met**	**Combined score**
	9	4	5	6	7	5	22
	4	6	8	8	6	4	24
	6	8	9	10	6	4	31
	6	7	6	8	6	3	24
	6	4	9	10	6	3	26
	6	6	10	10	5	3	30
	7	5	10	10	7	5	30
	5	4	6	6	7	3	17
	3	6	8	7	9	4	19
	4	5	10	8	4	4	27
	4	8	10	10	6	2	28
	7	3	10	10	4	4	30
	5	7	10	10	8	4	28
	6	8	9	10	7	6	32
	4	7	10	8	5	6	30
	4	5	9	10	5	4	27
	7	7	10	9	3	4	34
	6	3	6	6	5	4	20
Mean	5.5	5.72	8.61	8.67	5.89	4	26.61
Std deviation	1.5	1.67	1.72	1.57	1.49	1.03	4.74

Table 7: *Responses from users of the adjectival interface in the first stage for questions 1-6.*

	More time needed?	**Amount of extra time needed**	**Could decrease time required with practise?**	**Expected time per photograph**
	yes	10	yes	20
	yes	15	yes	10
	yes	5	yes	7
	yes	20	yes	15
	yes	60	yes	15
	no	*0*	yes	7
	yes	60	yes	30
	no	*0*	yes	7
	yes	10	yes	10
	yes	30	yes	10
	yes	60	yes	12
	yes	30	yes	30
	no	*0*	yes	10
	yes	20	yes	10
	yes	20	yes	40
	no	*0*	yes	10
	yes	20	yes	5
Mean		21.18		14.59
Std deviation		20.96		9.84

Table 8: *Responses from users of the direct specification interface in the first stage for questions 7 and 8.*

	More time needed?	**Amount of extra time needed**	**Could decrease time required with practise?**	**Expected time per photograph**
	no	*0*	yes	10
	no	*0*	no	*22*
	no	*0*	yes	15
	no	*0*	yes	10
	no	*0*	yes	15
	yes	10	yes	10
	no	*0*	yes	10
	yes	10	yes	10
	no	*0*	yes	12
	no	*0*	yes	5
	no	*0*	yes	5
	yes	10	yes	10
	yes	15	yes	10
	yes	10	yes	18
	yes	5	yes	15
	yes	10	yes	10
	no	*0*	yes	15
	no	*0*	yes	16
Mean		3.89		12.11
Std deviation		5.3		4.32

Table 9: *Responses from users of the adjectival interface in the first stage for questions 7 and 8.*

group in Tables 7 and 9. The data has been partitioned across the tables according to the types of questions to which they pertain: the data for questions 1 to 6 (those with answers on a scale of 1 to 10) are in Tables 6 and 7; the data for questions 7 and 8 (the time-related questions) are in Tables 8 and 9.

In addition, for the first partition of data we include a *combined score* — this is simply the sum of the responses to questions 1, 2, 3, 4 and 6 (where higher responses were better), minus the response to question 5 (where higher responses were worse). This gives an interesting metric of the user's overall experience. Furthermore, in the second partition of the data we include in italics implicit time values whenever the user responded with a "no" to the question and thus did not have to fill in a specific time value.

As can be seen from Tables 6 and 7, the means of the data from the adjectival group for the scale questions are higher than those of the data from the direct specification group. Looking at the time-related data, the means for the data from the adjectival group are lower than those of the data from the direct specification group. What remains is to determine whether these differences are statistically significant or not.

To evaluate whether the means of the adjectival group data are significantly larger than those of the direct specification group data, the hypotheses in Section 6.3 are utilised to establish null hypotheses. These can then be applied to a one-tailed Welch two sample t-test for each pair of sets of scale data, giving the results shown in Table 10.

#	**Null hypothesis**	*t*	*df*	*p*
1	$\mu_{\text{ADJ(accuracy 1)}} \leq \mu_{\text{DS(accuracy 1)}}$	1.9953	28.366	0.02785
2	$\mu_{\text{ADJ(accuracy 2)}} \leq \mu_{\text{DS(accuracy 2)}}$	1.8189	32.719	0.03905
3	$\mu_{\text{ADJ(easy to understand)}} \leq \mu_{\text{DS(easy to understand)}}$	7.5573	31.586	7.154e-09
4	$\mu_{\text{ADJ(easy to use)}} \leq \mu_{\text{DS(easy to use)}}$	3.1152	24.244	0.002338
5	$\mu_{\text{ADJ(frustration)}} \geq \mu_{\text{DS(frustration)}}$	0.9478	28.38	0.8244
6	$\mu_{\text{ADJ(expectations met)}} \leq \mu_{\text{DS(expectations met)}}$	0.275	28.693	0.3926
7	$\mu_{\text{ADJ(combined score)}} \leq \mu_{\text{DS(combined score)}}$	3.8195	24.632	0.000401

Table 10: *T-test results comparing the scaled data in Tables 6 and 7.* $\mu_{ADJ(x)}$ *indicates the mean of column x in the adjectival data;* $\mu_{DS(x)}$ *indicates the mean of column x in the direct specification data. The t, df and p columns give the t-value, degrees of freedom and p-value of the test, respectively.*

Employing a confidence level of 95%, these results indicate that in the first four cases and in the last case, the null hypothesis should be rejected in favour of the alternative hypothesis. That is, it is statistically likely that:

1. For the first photograph, users of the adjectival interface felt that their virtual landscape matched the photograph better than users of the direct specification interface.

2. For the second photograph, users of the adjectival interface felt that their virtual landscape

matched the photograph better than users of the direct specification interface.

3. Users found the adjectival interface easier to understand than users of the direct specification interface.
4. Users found the adjectival interface easier to use than users of the direct specification interface.
5. Users of the adjectival interface had a higher combined score than users of the direct specification interface.

In the case of the 5^{th} and 6^{th} t-tests, the p values lie outside the confidence level of 95% and so the null hypothesis cannot be rejected in either case. The p values are also not sufficiently large to consider the opposite hypotheses: namely, that the direct specification interface outperformed the adjectival interfaces in these cases. It is not possible, therefore, to conclude whether the expectations of the adjectival interface users were more successfully met than users of the direct specification interface, nor whether either group experienced a lesser degree of frustration.

Since users in this stage of the experiment performed two instances of the same task in sequence, it would also be interesting to analyse the differences in results of these two tasks. Significant differences in the data could indicate aspects such as a learning effect or user fatigue, either of which could influence the way in which other results are interpreted. To test for differences between the two tasks, we perform a two-tailed paired t-test on the adjectival and the direct specification data, giving the results shown in Table 11.

Null hypothesis	t	df	p
$\mu_{\text{DS(accuracy 1)}} = \mu_{\text{DS(accuracy 2)}}$	0.9871	16	0.3383
$\mu_{\text{ADJ(accuracy 1)}} = \mu_{\text{ADJ(accuracy 2)}}$	0.3688	17	0.7168

Table 11: *T-test results comparing the data for the two tasks given to users.*

In both the adjectival and direct specification t-tests, the p value lies outside the confidence level of 95% and so the null hypothesis cannot be rejected. It can therefore be concluded that there is no statistical difference in the means between the first and second tasks for either group, and therefore that no learning effect or user fatigue can be inferred from these results.

The fact that the two tasks do not exhibit statistically different results allows an additional comparison between the adjectival and direct specification groups to be made: since the two tasks performed were the same, the results from both tasks can be treated as having been drawn from a single task. The summary of this combined data is given in Table 12, where again the adjectival interface has a higher mean.

As before, a one-tailed Welch two sample t-test is performed to test the significance of the difference, which gives the result shown in Table 13.

With a confidence interval of 95%, the null hypothesis is rejected and one can conclude that the mean accuracy for the adjectival interface is indeed higher than that of the direct specification interface.

Interface	# samples	Mean	Std deviation
Direct specification	34	4.5	1.83
Adjectival	36	5.61	1.57

Table 12: *Summary of combined accuracy data, which treats the two tasks performed by users as an instance of the same, common task.*

Null hypothesis	t	df	p
μADJ(accuracy combined) $\leq \mu$DS(accuracy combined)	2.7175	65.188	0.004208

Table 13: *T-test result comparing the combined accuracy data of the adjectival interface against that of the direct specification interface.*

In the same fashion, one-tailed t-tests can be performed on the time data to determine whether or not the means for the adjectival group are indeed lower than the means for the direct specification group. The results of these t-tests are given in Table 14.

Null hypothesis	t	df	p
μADJ(extra time) $\leq \mu$DS(extra time)	3.303	17.931	0.001986
μADJ(expected time) $\leq \mu$DS(expected time)	0.9549	21.692	0.1751

Table 14: *T-test results comparing the time data of the adjectival and direct specification groups.*

With a confidence level of 95%, the null hypothesis for the first test can be rejected, inferring that on average users of the adjectival interface felt they needed less time to perform their task than users of the direct specification interface. One cannot, however, reject the null hypothesis in the second test, and so cannot draw any conclusions about how long users thought they would take to complete the task after sufficient practise in using the system.

6.4.2 Second stage

Although the analysis of the data from the first stage resulted in some favourable conclusions for the adjectival interface, they are largely based on qualitative data and do not address the issue of which interface leads to more faithful content generation. For the second stage, 89 participants each provided 15 data points, in the form of choosing which of a pair of virtual landscapes more faithfully captured a particular photograph. This gives a total of 1335 data points — 566 of which were in favour of landscapes generated using the direct specification interface, and 769 in favour of those generated with the adjectival interface.

Again, it needs to be determined whether this distribution occurred by chance, or if the adjectival interface does perform statistically better than the direct specification interface. Consider the null hypothesis that these results were generated by a random and fair process. Since there are exactly two possible choices for each data point, this is the canonical Bernoulli process [Helstrom, 1984] in which the probability of making either choice is 0.5. The probability of this random process choosing the adjectival interface at least 769 out of 1335 times can be ascertained by consulting the binomial

distribution. Using the binomial test of GNU-R [R Development Core Team, 2007] to conduct a two-tailed test, this probability is found to be $3.045e-08$. This is well within a confidence level of 95%, and so the null hypothesis can be rejected.

The rejection of the null hypothesis indicates that the data collected could not have been the result of a random and fair process. The actual probability of choosing, say, the adjectival interface, is then not equal to 0.5. Using the observations of this stage of the experiment, the probability is easily calculated as $\frac{769}{1335} = 0.57603$, which indicates that users are more likely to choose images generated by the adjectival interface. This was, however, calculated from a relatively small sample of responses, and is not entirely representative of the true underlying probability. Fortunately, the binomial test offers a means for also reasoning about the true probability of the process. Again using GNU-R with a confidence level of 99.9%, it can be found that the true probability for the data observed lies in the interval $[0.5308152; 0.6203767]$. That is, if one ran many random simulations — using different probabilities for choosing the adjective interface — and kept aside all those that resulted in the adjectival interface being chosen 769 times out of 1335, then in 99.9% of these cases the underlying probability would be in the range $[0.5308152; 0.6203767]$. That is, in 99.9% of cases, users will be more likely to prefer content generated with the adjectival interface, over content generated with the direct specification interface.

Of additional interest is whether any photographs were particularly favoured by the users. Table 15 shows the distribution of responses associated with each photograph, and the resulting p-value from a two-tailed binomial test.

Again, all of the p values are well within a confidence level of 95%, and so for each photo we can conclude that the true probability of choosing the adjectival interface is not equal to 0.5. It should be noted, though, that for the third photograph users preferred the *direct specification* interface, and so in this case the true probability of choosing the adjectival interface is less than 0.5. For each of the other photographs, though, the adjectival interface was preferred, and for these the true probability of choosing the adjectival interface is thus greater than 0.5.

6.5 Interpretation of results

The key goal behind an adjectival interface is to provide an easy-to-use and intuitive means for users to create compelling procedural content. The results in Tables 6 and 7 provide the first evidence that this has been successful — in almost all cases, the adjectival interface not only scored better than the direct specification interface, but also exhibited lower standard deviation values. The latter point is important as it indicates greater consistency amongst users' responses which, in combination with the higher mean scores, suggests that the adjectival interface will appeal to and be usable by a wider range of users than the direct specification interface. Some users may, of course, prefer the additional control provided by direct specification of parameters, but since the adjectival interface is designed with a full procedural system "under the hood", such users can easily strip away the

Photograph	Direct specification interface	Adjectival interface	p
	115	152	0.02740
	92	175	0.0000004247
	155	112	0.01003
	96	171	0.000005187
	108	159	0.002155

Table 15: *Results of second stage experiment grouped by photograph.*

adjective interface if they so desire.

The statistical analysis of the first stage results — summarised in Table 10 — further strengthens the support for the adjectival interface, by showing that the majority of the difference in the observed mean values were in fact statistically significant. Although users of the adjectival interface reported higher frustration levels, this was not a significant result, and so cannot be counted against the adjectival interface. The fact that neither group showed a statistically significant difference in the level to which their expectations were met, can also be seen in a positive light — it shows that an adjective interface has not diminished the expectations of users.

With the first stage clearly highlighting that the adjective space is preferred by users, what remains to be seen is whether or not the user's ability to create compelling content has been jeopardised.

It would suffice here if the two interfaces performed equally well — since then one would have an interface that is easier to use, and which produces comparably suitable content.

The results of the second stage, however, go beyond this requirement by showing that the adjectival interface actually performs better than the direct specification interface for the task at hand. What is more, the 99.9% confidence interval of $[0.5308152; 0.6203767]$ suggests that this is a marked difference — that is, 99.9% of the time the adjective interface will perform better, generating content that is deemed more accurate than that of the direct specification interface in at least 53% of all cases.

Evaluating the results of the second stage on a per-photograph basis, however, demonstrates that the adjectival interface is not entirely superior. For the third photograph, users preferred images generated with the direct specification interface. It is not immediately obvious why this is the case — it is possible that with more training data, the adjectival interface might perform better and ultimately produce more compelling output.

6.6 Further evaluation

Although the experiment presented has produced positive results, this was for one particular procedural system and does not demonstrate the generality of the adjectival interface. To show how the technique generalises, it has been applied to two other procedural domains. No formal user experiments have been run; however, the informal responses from users have been positive and the applications are presented here to demonstrate that the technique can be generalised to other domains, with varying numbers of parameters and degrees of complexity.

6.6.1 Tree generation

In Chapter 2, mention was made of the tree generation technique of Weber and Penn [1995] which requires 80 parameters to control the generation of a single tree. The high number of parameters makes the technique well-suited to simplification via an adjectival interface. The parametrisation is also challenging in that there is a good deal of interaction between the parameters. For example, one parameter controls the number of levels of recursion which are performed in the process of creating the trunk and branches, and several groups of parameters define how the behaviour is adjusted for each level. If only a single level of recursion were used, there would therefore be a large number of parameters whose values would be irrelevant as they would never be referenced in creating that tree instance.

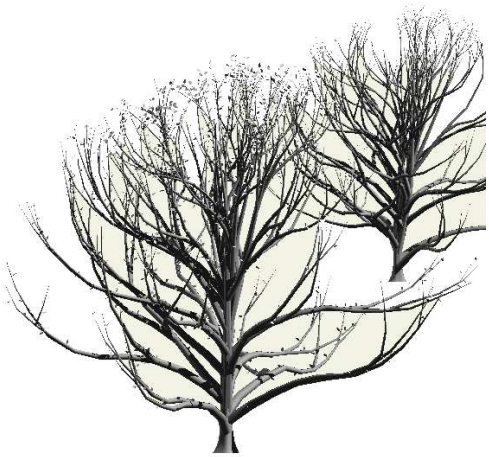
A total of 750 trees were used to train the RBFNs for this system, using 20 adjectival descriptors.

Examples of trees generated with this system and the adjectives used to generate these trees are shown in Figure 51.

(a) Described as wide, lush, tapered, top-heavy, *not* skeletal and *not* drooping.

(b) Described as bending, branching, warped, *not* lush, *not* tropical and *not* majestic.

(c) Described as tropical, straight, top-heavy and drooping.



(d) Described as lush, wide, *not* skeletal and bending.

Figure 51: *Some examples of trees generated using an adjectival interface, utilising the procedural framework of Weber and Penn [1995].*

Whilst the adjectival interface produced many adequate trees (such as in Figures 51(a) and 51(c)), the results did not correctly follow the adjectival descriptors as successfully as in the landscape generation experiment (see for example Figures 51(b) and 51(d)). One possible reason for this could be that the larger parameter space of Weber and Penn's model requires significantly more sampling during the training phase. Related to this might be the complexity of the parameter space, whereby large regions of the parameter space are "redundant" due to the interaction between the parameters in this model. Means for ameliorating this are presented and discussed in Chapter 7.

6.6.2 Emotive speech synthesis

The concept of emotive speech synthesis was also mentioned in Chapter 2, specifically the work of Oudeyer [2003], in which the author presents an algorithm for the generation of meaningless baby-like speech subject to 10 procedural parameters. Due to the "speech" not having any intrinsic meaning, the focus of the algorithm is on imparting various emotional characteristics such as anger, happiness or sadness, without having to deal with the much harder problem of producing correct speech.

Oudeyer's algorithm employs 10 parameters, 7 of which are real-valued and 3 of which are categorical, and with no complex parameter interaction like that of Weber and Penn's tree generation technique. As such, only 80 samples were used for the training of RBFNs, and 9 adjectival descriptors were used to describe each sample.

The speech generated through the adjectival interface was typically well matched to the descriptors used, as one might expect for such a small number of procedural parameters. Varying emotions in speech can tend to be quite similar, which may also have contributed to the good results — in that distinct emotions can be hard to properly distinguish, and often rely on the semantic information being conveyed to be properly categorised. As such, a sound that conveyed one emotion could quite easily also be seen as conveying several other, distinct emotions during the training phase. The use of WordNet for semantic information was likely instrumental in the success of an adjectival interface for this application, as the semantic links provided allow for the emotional ambiguity of speech to be overcome by considering many related adjectival descriptors.

Unfortunately, due to the medium of sound, we cannot produce any examples here — but the interested reader may find examples for download at:

`http://people.cs.uct.ac.za/~chultqui/speech_samples`

6.7 Summary

In this chapter, an experiment for testing the effectiveness of our adjectival interface was presented and the results of the experiment disclosed. Some statistical analysis of the data was performed, from which it was concluded that adjective space is not only preferred by users, but also produces content more accurately than an interface where the user has direct control over the underlying procedural parameters. It was also demonstrated that an adjectival interface can be successfully applied to other procedural domains.

Chapter 7

Conclusions and future work

At the outset of this thesis, a technique was sought that provided the following three features:

1. Allow large and complex procedural content to be created quickly.
2. Provide an interface that is usable by novice and non-technical users.
3. Maintain the flexibility afforded by parametrised procedural models.

Although it has been explained how an adjectival interface would satisfy these requirements, it is worth revisiting these points again, armed with the full knowledge of the technique.

As Chapter 2 demonstrated, the field of procedural modelling is well-researched and continues to evolve and improve. Progressively more complex content can be generated in decreasing amounts of time, in part due to the acceleration of hardware, but also due to the increased sophistication of the procedural techniques available. As such, the problem of quickly generating large and complex content is already solved, through the harnessing of procedural methods "under the hood" by the adjectival interface.

In Chapter 1, it was noted that a key problem with technology was that humans are unable to keep up with the rate of technological development. This is closely tied to the concept of providing an interface that is usable by novice and non-technical users — if a novice user is able to easily make use of the latest technology through an intermediate interface, then the problem of humans keeping up with technology is no longer an issue. As humans communicate using language, the overhead for a user to adapt to an adjectival interface is arguably much less than for a parametrised interface — which is verified by the results presented in Chapter 6.

Finally, by employing procedural techniques for the actual creation of content, there is no sacrifice in flexibility. One could argue that the adjectival interface on its own is less flexible, but the goal

was not to replace procedural techniques, and rather to augment them with a more usable interface — in this, the adjectival interface presented is a success. Chapter 6 showed that users were able to generate content with an adjectival interface that more faithfully fulfilled a specific task, which in itself is fair proof that no flexibility has been sacrificed. Furthermore, since the adjectival interface is implemented as a layer on top of procedural techniques, it does not preclude a user from stripping away the adjectival interface and interacting directly with the procedural parameters. The adjectival interface can thus serve as a form of intrinsic scaffolding, or be used for a rapid approximation that is later refined through adjustments to the underlying procedural parameters.

7.1 Motivational summary

Having re-iterated how the final technique addresses the goals of this thesis, we now summarise the motivations that guided the direction of this research, and which culminated in the final system presented.

Having identified that procedural models are immensely powerful but unwieldy to use, an alternative interface to those models was sought. By modelling the interface as an extra layer of abstraction above the procedural models, the problem of modelling is reduced to finding a suitable mapping from adjectival descriptors to procedural parameters — this was discussed in Chapter 3. By restricting the focus of the thesis to parametrised procedural models, and using scalar values to represent varying degrees of quantification for adjectival descriptors, finding this mapping was shown to be equivalent to finding a multi-dimensional function approximation. Various techniques for finding such an approximation are available, and these were discussed in Chapter 4. The specific application of function approximation to an adjectival interface was the focus of Chapter 5, and a suitable function approximation scheme was chosen. Domain-specific problems were also noted and appropriate solutions developed; these included:

- Identifying that the approximation of the inverse function, f^{-1}, was more suitable for the mapping between adjective space and parameter space, and determining appropriate methods for dealing with the inverse.
- Developing suitable methods for representing adjectives in a format suitable for numerical function approximation.
- Providing for a dynamic adjective space, by the novel extension of radial basis function networks to include *certainty values*, thus allowing for easy inclusion of the semantic network information provided by WordNet.

Finally, in Chapter 6, the overall implementation was tested and evaluated. An experiment was designed to compare and contrast the effectiveness of the adjectival interface, against an interface allowing for direct specification of the procedural parameters. Both qualitative and quantitative

comparisons were made through the use of two stages in the experiment — in the first stage, participants were divided into two groups, and each group was assigned either the adjectival or direct specification interface. Participants used their designated interface to complete a task, after which they were asked questions that provided a qualitative measure of their experience. On almost all counts, the adjectival interface was found to provide a qualitatively better experience than the direct specification interface.

The second stage then provided a quantitative comparison, by having a separate group of participants compare pieces of content generated using the two interfaces, and decide which piece of content had performed better at the task given to participants in the first stage. In this respect, it would have been sufficient if the adjectival interface were shown to be just as good as the direct specification interface. The results surpassed this goal, however, and it was found that participants preferred content generated using the adjectival interface.

7.2 Future work

Although the final experiment conducted for this research had a positive outcome, there are still areas in which this technique could be improved and further research conducted. Some of these are now briefly summarised and possible avenues of specific exploration suggested.

7.2.1 Better handling of non-continuous procedural methods

In Chapter 5, we posited that two sets of procedural parameters which are very similar should elicit similar adjectival descriptions. Whilst this is true of many procedural systems, and is in fact desirable from a procedural system so that users have some modicum of control over the output, it is not a requirement or guarantee. Better handling for such systems would be valuable research — if one had knowledge of where these discontinuities occurred in parameter space, then one could perhaps partition parameter space and employ our method on several continuous sub-regions.

7.2.2 Conduct experiments with a more focused group of users

In this thesis, the experiments were carried out with a group of “beginners”. Whilst the system is designed to support novice and non-technical users, there is surely a class of users who lack the requisite technical or mathematical abilities to work with procedural parameters, but who might make frequent use of an adjectival system. However, finding such a specialised user group is difficult. It would be interesting to find a sufficiently large group of such users, such that the experiments in this thesis could be repeated. This would then give results that more accurately pertain to those who would use an adjectival system.

7.2.3 Hybrid personalised function training

In Chapter 5, we discussed several means for eliciting data with which to train the function approximation. In general, having each user fully train a personalised approximation may be too cumbersome, yet having a single expert or a synthesised opinion form the basis for the training might not suit every individual user.

Although it might be assumed that different users have different perceptions, our preliminary investigations[1] indicate that people in general have some fairly similar perceptions. This is an area that has been well researched in psychology, and in particular language has been shown to have a great impact on shaping people's perceptions. Davidoff et al. [1999], for example, demonstrates that memory and perceived similarity of colours are predicted by the colour terms in a speaker's language.

As such, one could imagine a system in which an expert or synthesised opinion forms the basis for the training of the function approximation, subject to some small subset of per-user training to personalise the approximation. One easy way in which this could be accomplished using the framework of this thesis was described in Section 5.6.1, in which it was suggested that certainty values could be used for exactly this purpose. Using such an approach has the benefit of maintaining both the common and individualised data, and seeking a best fit where the user data has a slightly greater weighting.

7.2.4 Direct comparison to other state of the art interfaces

The experiments presented in this thesis were aimed at showing that an adjectival interface was not only usable, but also provided an advantage over directly controlling procedural parameters. Whilst a more direct comparison to other state of the art interfaces could have been performed, this would have restricted the problem domain being addressed — for example, much work has been done in producing visual interfaces for computer graphics content, but such interfaces are not useful when generating procedural speech.

Now that an adjectival interface has been shown to have promise, it would be interesting to conduct more direct comparisons with state of the art interfaces in specific problem domains. One could even imagine a hybrid approach — for example, an adjectival interface could be used to select a number of virtual environments, which can then be explored using a design gallery [Marks et al., 1997].

7.2.5 Online training

Closely related to the previous point is the topic of online training. Here, the user could train the system as they use it — if, after generating content, they felt that the content did not match the

[1] See Appendices A and B

descriptors that they had used, they would then have the opportunity to specify how the content should be described and, in so doing, further train the system. Similarly, if the user was satisfied with the content generated, then this information could be used to strengthen some of the underlying approximation controls.

7.2.6 Local control

The technique proposed in this thesis offers only global control of content generation — the adjectival descriptors apply to the content as a whole, and it is not possible to differentiate the behaviour on a finer scale. For example, consider the task of landscape generation presented to the users in Chapter 6: although the landscape might have a suitable overall look and feel, it would be extremely useful to be able to specify individual characteristics for particular regions of the landscape.

This is not an easy problem to solve, and could be seen as being very specific to each particular procedural domain. One avenue of exploration would be to define different sets of procedural parameters at different points in space or time — depending on the nature of the underlying procedural system — and then smoothly interpolate between these. An alternative idea would be to first generate content using a global description that captures the overall feel, and then somehow hone in on specific areas to apply local modifications.

7.2.7 Natural language parsing

Whilst the use of adjectival descriptors has proven to be successful, it would be even more intuitive for users if they could describe a scene using natural language. WordsEye [Coyne and Sproat, 2001] could perhaps be extended to include the notion of a procedural model in its database, thus allowing for the mapping from nouns to particular procedural models. The adjectives used in conjunction with each noun could then be applied to the procedural model using a technique similar to that of this thesis.

7.2.8 Use of non-parametric procedural inputs

In Chapter 2, a wealth of procedural techniques was discussed. Several of these use inputs that are not strictly parametric, such as image maps or sketches. It would be interesting to consider the integration of such techniques into an adjectival framework — perhaps by having the user supply input in some other form and then modify that input based on an adjectival description.

As an example, consider the landscape generation system of Chapter 6. If a user wanted finer control over the heightmap used, it would be a relatively straightforward task to have the procedural system skip its internal heightmap generation and simply use the user-provided heightmap instead. The

adjectival interface could then be employed to perform the remaining tasks, such as landscape shading and the creation of rivers, lakes and trees.

One could also conceive of a procedural system that takes as input a heightmap or a sketch, and warps or modifies that input in some way based on a set of procedural parameters. An adjectival interface could be employed in such a setup to control the modification process, which can then be applied to other alternative forms of input.

7.2.9 Incorporating pre-defined content

Closely related to the previous point, is the notion of incorporating some pre-defined content and having the adjectival interface generate additional, augmenting content. With respect to landscape generation, for example, one could conceive of a user wanting to place several buildings onto the landscape, using existing architechtural models, and having the landscape evolve around the buildings. This is in some ways an extension from supporting non-parametric procedural inputs, to more generally supporting alternative forms of input that in some way interact with the procedural generation.

Whilst this seems like it would largely require adaptation of the procedural model used — adapting the model to cater for pre-defined content — it does tie into the adjectival realm, in that the pre-defined content might *itself* confer some adjectival properties. Consider the example of placing a house on a landscape, where the landscape is generated using an adjectival interface. If the user wished for a "flooded" landscape, care would need to be taken to ensure that the house nonetheless is placed on a piece of solid land. A wooden house in the middle of a desert also seems improbable; however if that house were situated next to an oasis, that might be more plausible.

In general, therefore, one could imagine some sort of feedback mechanism whereby the pre-defined content is able to affect the adjectival description of its locality — something akin to open L-systems [Měch and Prusinkiewicz, 1996] which allow for communication between the procedural model and its environment.

7.3 Final thoughts

As technology advances, progressively more incredible feats can be performed using a typical home computer. Yet, without the proper tools to harness this power, the average user is totally oblivious to the magic that is mere keypresses away. It is the responsibility of computer scientists to share that magic and make it more accessible, and it is our hope that this thesis has shown one means in which that process of sharing could be accomplished.

Appendix A

First investigation: adjective representation and consistency of thought

One of the key characteristics of a descriptive method for the generation of procedural content is how the adjectives are represented. Of additional importance to the complexity of the final system is how similar people's perceptions are, as high similarity could lead to a simplified mathematical model. The first user investigation aimed at addressing these two issues.

A.1 Outline of investigation

A simple procedural environment was defined with 16 procedural parameters, which are shown in Table 16. For simplicity, the environment was constrained to be an island in the middle of the sea, with a limited selection of tree types and no additional foliage. With this investigation being intended as very prototypical and solely for the purpose of gathering some more general information, the focus was not on high levels of realism. This also allowed for a real-time rendering, giving users the opportunity to explore the environment at their will instead of being constrained to discrete images. Some examples of the types of environments generated are shown in Figure 52.

With a system in place for generating environments procedurally, 15 environments were chosen as exemplars of the diversity achievable by the system. The values of the procedural parameters used for these 15 environments are shown in Table 17.

15 test subjects were solicited and given the opportunity to explore the 15 environments, and describe them. Before interactively exploring and describing any of the environments, each user was first

(a)

(b)

Figure 52: *Some procedural environments generated for our first investigation.*

Parameter	Description	Min	Max	Default
time-of-day	The hour part of the time of day	0	23	12
perlin-increment	Inverse frequency on Perlin noise used to create height values for the terrain.	0.0001	1.0	0.01
perlin-max	Maximum height value of Perlin noise.	0.0	300.0	150
min-gaussian	Minimum Gaussian output.	0.0	1.0	0.0
max-gaussian	Maximum Gaussian output.	0.0	1.0	1.0
num-gaussian	Number of Gaussians to blend with Perlin noise in creating terrain.	5	60	20
min-radius	Minimum radius of Gaussian landscape components.	25	225	25
max-radius	Maximum radius of Gaussian landscape components.	25	225	225
min-rainfall	Minimum amount of rainfall that falls on the terrain.	0.0	1.0	0.0
max-rainfall	Maximum amount of rainfall that falls on the terrain.	0.0	1.0	1.0
rainfall-variance	Inverse frequency of rainfall variance over the terrain.	0.0	1.0	0.25
palm-density	Proportion of landscape locations that should have palm trees.	0.0	1.0	0.0005
cactus-density	Proportion of landscape locations that should have cactii.	0.0	1.0	0.0005
fir-density	Proportion of landscape locations that should have fir trees.	0.0	1.0	0.0005
cloudiness	Proportion of the sky that is covered by cloud.	0.0	1.0	0.5
rain-threshold	Minimum amount of cloud-cover required to generate rain.	0.0	1.0	0.7

Table 16: *Procedural parameters for our first investigation.*

shown a set of images captured as single-frame screenshots of the environments — the purpose being to make the user aware of the diversity in the system, serving as a guide in their descriptions. After interactively exploring an environment, the user was asked to describe their environment using a *slider interface*, as shown in Figure 53. Seven adjectival descriptors were provided, which represented either a single adjective or an antonym-pair of adjectives (for example "wet/dry"). These descriptors were chosen to reflect the visual and affective differences in the environments shown. Each descriptor had an associated slider control to indicate its relevance — as the slider was adjusted, the label associated with the slider would be updated to reflect both a numerical value and a categorical description corresponding to the slider position. The categories were defined by a simple partitioning of the numerical range into equal-sized partitions, with 5 categories per adjective (giving 10 categories for antonym-pair descriptors). User comments on individual environments were also collected, as well as general comments once all the environments had been viewed and described.

	Environment #														
Parameter	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
time-of-day	12	14	16	19	5	7	7	11	13	17	10	18	0	18	8
perlin-increment	0.01	$5e^{-3}$	$5e^{-3}$	0.01	0.01	0.01	0.01	0.01	0.01	$5e^{-3}$	$2.5e^{-3}$	0.02	$1e^{-3}$	0.01	0.01
perlin-max	150	50	30	150	150	200	200	75	300	200	100	100	100	300	50
min-gaussian	0	0	0.7	1	0	0	0	0	1	0.6	0.6	0.6	0	0.6	0.6
max-gaussian	1	0.6	1	1	1	1	1	0.6	1	1	1	1	1	1	1
num-gaussian	20	20	40	30	30	30	30	40	40	40	40	20	20	40	15
min-rainfall	0.5	0.6	0.6	0	0	0.8	0	0	0.8	0.8	0	0.6	0	0.8	0
max-rainfall	1	1	1	1	1	1	0.3	0.3	1	1	1	1	1	1	0.3
rainfall-variance	0.25	0.25	0.25	0.5	0.1	0.1	0.1	0.1	1	1	1	0.25	0.5	1	0.25
palm-density	$3e^{-3}$	$1.5e^{-4}$	$1.5e^{-3}$	$1.5e^{-3}$	$1.5e^{-3}$	$1.5e^{-3}$	$1.5e^{-3}$	$1.5e^{-3}$	0	$4e^{-4}$	$4e^{-4}$	$5e^{-4}$	$1.5e^{-3}$	$4e^{-4}$	$3e^{-3}$
cactus-density	$5e^{-4}$	0	0	0	0	0	$1e^{-4}$	$1e^{-4}$	0	0	$1e^{-4}$	0	0	0	0
fir-density	$7.5e^{-4}$	0	0	$1e^{-4}$	$1e^{-4}$	$1e^{-4}$	$1e^{-4}$	$1e^{-4}$	0	$1e^{-5}$	$1e^{-4}$	$5e^{-4}$	$1e^{-4}$	$1e^{-4}$	0
cloudiness	0.5	0.1	1	0	0.8	0.8	0	0	0.9	0.2	0.2	0.9	0	0.9	0
rain-threshold	0.7	0.7	1	1	0.8	0.8	1	1	0.7	0.3	0.3	0.7	1	0	0.7
min-radius	25	25	25	100	100	100	100	25	100	25	25	25	100	25	25
max-radius	225	40	30	100	100	125	125	30	200	200	70	75	150	200	225

Table 17: *The parameter values used to generate environments for our first investigation.*

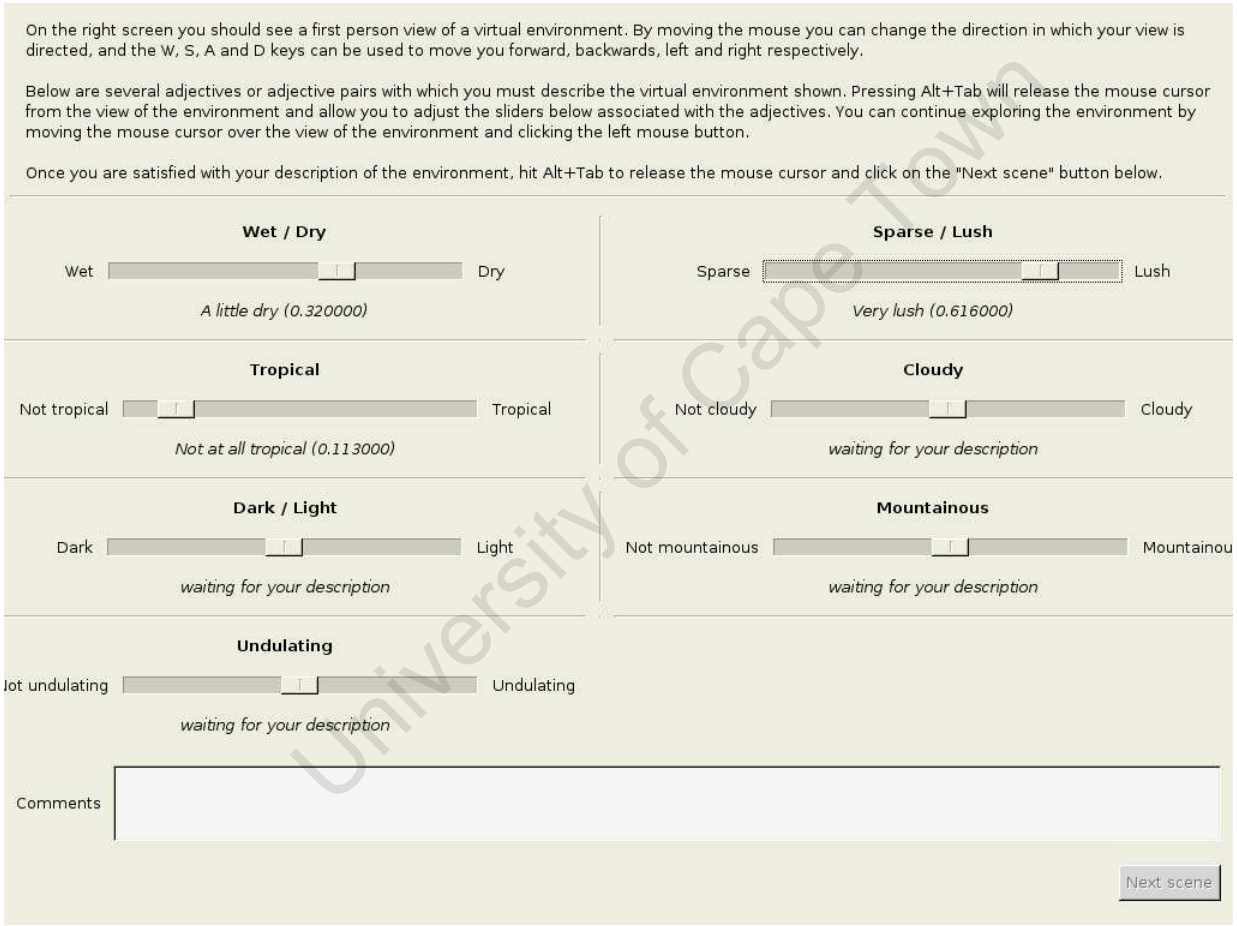


Figure 53: *The user interface for capturing descriptions in our first investigation.*

A.2 Objectives

The purpose of this investigation was threefold and sought to explore the following areas of interest:

1. **Adjective representation.** By providing a slider interface with associated dynamic labels as described, the user is given the option of either thinking about their descriptions numerically, or categorically. Through user comments and discussion with the users afterwards, it was hoped to establish whether the users found the numerical interface useful, or whether they preferred a categorical interface. The inclusion of antonym-pairs of adjectives also provided a means for assessing the usefulness of these, or whether it would make more sense to separate out the pairs into individual descriptors.

2. **Common user perceptions.** As all users would give descriptions for the same 15 environments, this should allow for the analysis of similarities and degrees of disparity in their descriptions.

3. **Initial testing and analysis of function approximation techniques.** Although typical function approximation schemes require large amounts of data in order to obtain accurate approximations, smaller quantities of data can be useful in generating a coarser approximation. The data collected for each user can be used after the investigation to train and test function approximation techniques, in an effort to examine how well the techniques perform with small quantities of training data and ascertain whether a larger training corpus is required for this problem domain. If a small training corpus was sufficient to train an approximant, then this would allow for easy customisation of the approximants to each individual user, by allowing the user to train the approximant themselves.

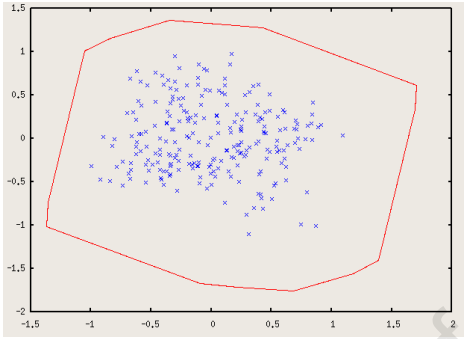
A.3 Results and discussion

After running the investigation and analysing the data, the following observations were noted:

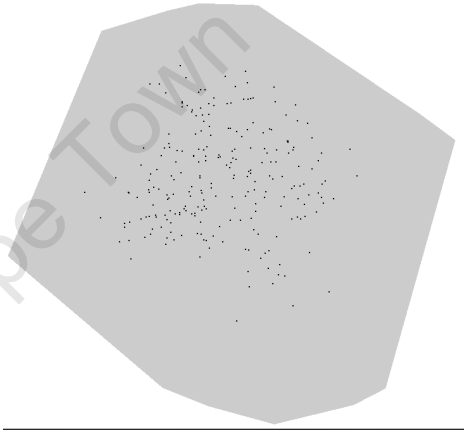
1. **Users preferred the categorical interface for selecting adjectival descriptors.** Whilst some users made full use of the interface provided by utilising both the categorical and numerical indicators — first choosing an appropriate textual category and then fine-tuning their selection using a numerical value within that category — for most users the slider interface was cumbersome and confusing. The general consensus amongst users was that by the third or fourth environment, they would adjust the slider solely based on the textual category that was displayed, totally ignoring the numerical display.

2. In considering the similarities in users' perceptions, we first discuss how such a criterion might be objectively evaluated. In a geometric sense, if the users gave similar descriptions for one environment then one would expect these data points to be clustered together in adjective space.

Such spatial analyses are often best achieved through a human visualising the data, so as an initial metric we consider some graphical representations. This in itself is not straightforward due to the seven-dimensional adjective space employed in the investigation, but by considering a reduction in dimensions using principal components analysis, a good approximation to the data can be obtained. Two and three -dimensional reductions of the data collected can be seen in Figures 54(a) and 54(b). The three principal eigenvalues resulting from the reduction were 0.494215, 0.218662 and 0.167874, which indicates that the two-dimensional figure represents the data with an accuracy of 71.3%, and the three-dimensional figure with accuracy of 88.1%.



(a) Two-dimensional reduction of the data. The red line indicates the boundary of legal data values.



(b) One view of a three-dimensional reduction of the data. The grey area indicates the 3D convex hull of legal data values.

Figure 54: *Visualisations of the data collected for the first investigation, making use of principal components analysis for dimensionality reduction.*

The dimension-reduced plots in Figures 54(a) and 54(b) also do more than show us where the data-points lie in relation to each other — they also show us how the points are positioned relative to adjective space as a whole. From this we can observe that for these 15 environments, adjective space was not well sampled as there are no observations close to the boundaries of the space, and most of the points are clustered just slightly away from the centre of adjective space. This indicates that people's perceptions were in general correlated (as there are very few significant outliers), but does not clearly indicate how closely their descriptions matched on a *per-environment* basis.

If one limits the diagrams to show only the descriptions associated with each individual environment, then a much clearer picture of the clustering involved is achieved. These plots can be found in Figures 56, 57, 58 and 59 at the end of this chapter.

Although these visualisation help in better understanding how the data is grouped, they don't really help to make any factual statements about the clustering of the data. One could, for example, say that the data for environment 12 is particularly well clustered, but with reference to what? An objective means of measuring the inherent clustering of the data is required.

K-means clustering [MacQueen, 1967] is a technique that is often used for the geometric clustering of points, and seeks to minimise the summed distance from each point to its cluster's centroid. K-means is a particularly good choice of clustering solution because of the strong dependence between a cluster's centre and each point's membership in a cluster — the cluster centre is defined as the centroid of all the cluster's members, and for a point to be in a cluster it must be closer to that cluster's centroid than any other cluster. One could, therefore, use K-means to create 15 clusters from the data of this investigation, with one cluster per environment. A means for measuring the relative strength of the clustering is still required; some options are:

1. **Correct cluster on termination.** K-means is an iterative algorithm, which operates by starting with one clustering and iteratively trying to improve upon it. The initial clustering given to the algorithm is typically random, although one alternative is to prime the algorithm with a particular state. In this case, one could start with the initial state being what is actually desired — namely that the resulting clusters are exactly the different subsets of points associated with each environment. After the K-means algorithm has terminated, one can then examine how many points have stayed in their initial clusters.

2. **Common cluster members.** One problem with the metric above is that it does not take into account the possibility that a number of points from one cluster could "migrate" to another cluster during the execution of the algorithm. So although the final clustering might be good, it would score poorly due to points not having stayed in their original cluster. One way around this is to count, for each point, the number of other points in the same cluster that described the same environment.

To obtain an objective idea of how good each of these metrics is, the results obtained are compared to those obtained on average from a number of random datasets. Table 18 describes these results.

	Experimental data	**Average for random data**	**Minimum for random data**	**Maximum for random data**
Correct cluster on termination	51.25%	12.72917%	8.75%	15.4167%
Common cluster members	35.7778%	7.24722%	6.11111%	9.05556%

Table 18: *A comparison between the experimental data in the first investigation, and random data, for various clustering metrics. All random data used had the same number of points as the experimental data, so as to ensure a fair comparison.*

As can be seen in Table 18, the data gathered in the investigation performs significantly better than random data for the metrics described, indicating that there is some degree of clustering as was expected.

If that were not sufficient evidence, consider one more form of verification from a slightly different angle. It is reasonable to say that in a well clustered dataset, the average distance from a point to

others describing the same scene should be less than the average distance from that point to others describing different scenes. So if one computes the ratio of those two averages, this would give a value describing how much closer a point is to other points describing the same environment, as compared to points describing different environments.

More formally,

$$r_i = \frac{\frac{\sum_{\mathbf{p}\in E_i} \|\mathbf{p_i}-\mathbf{p}\|}{|\{\mathbf{p}:\mathbf{p}\in E_i\}|}}{\frac{\sum_{\mathbf{p}\notin E_i} \|\mathbf{p_i}-\mathbf{p}\|}{|\{\mathbf{p}:\mathbf{p}\notin E_i\}|}} \tag{15}$$

is computed for each point $\mathbf{p_i}$, where E_i is the set of all other points describing the same environment as p_i. Taking an average of all the r_i values then gives a metric for the clustering of the dataset as a whole. Intuitively, one could see that for random data the average r_i should be approximately equal to 1 — checking this empirically with several random datasets, this is indeed found to be the case. A comparison of random data to that collected in the first investigation, using the ratio described, is given in Table 19.

Experimental data	**Average for random data**	**Minimum for random data**	**Maximum for random data**
0.447186	1.00342	0.984731	1.02468

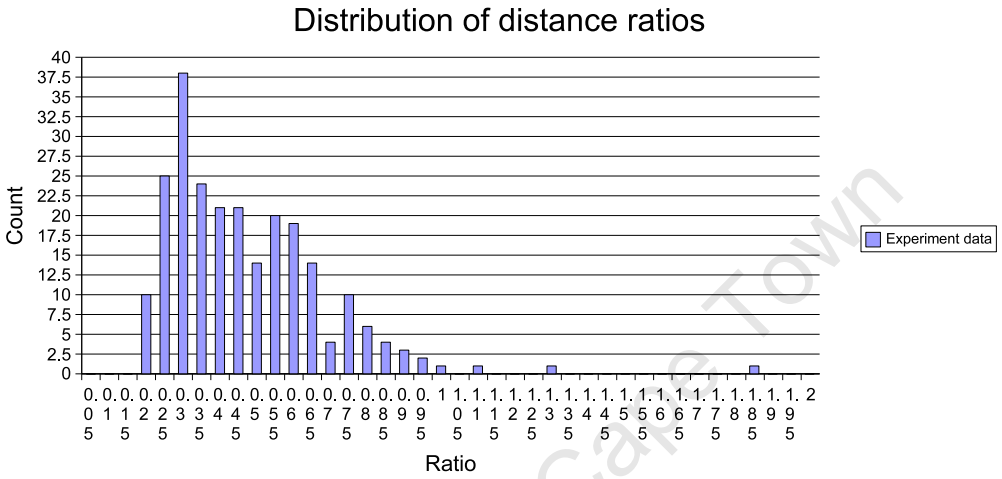
Table 19: *A comparison between our experimental data and random data, for the average ratio of distance to points describing the same environment over distance to points describing different environments. 1000 random data sets were sampled, and the minimum and maximum values refer to the minimum and maximum average r_i values observed over those data sets.*

Another way of analysing this metric is to consider a histogram of the ratios r_i: the histogram for the experimental data and that of a random data set are shown in Figure 55. As can clearly be seen in these histograms, the ratios for random data exhibit the characteristics of a narrow normal distribution with mean 1 (as one might expect), whilst the majority of the experimental data shows ratios significantly less than 1, with only a few cases having larger ratios that could be classified as statistical outliers.

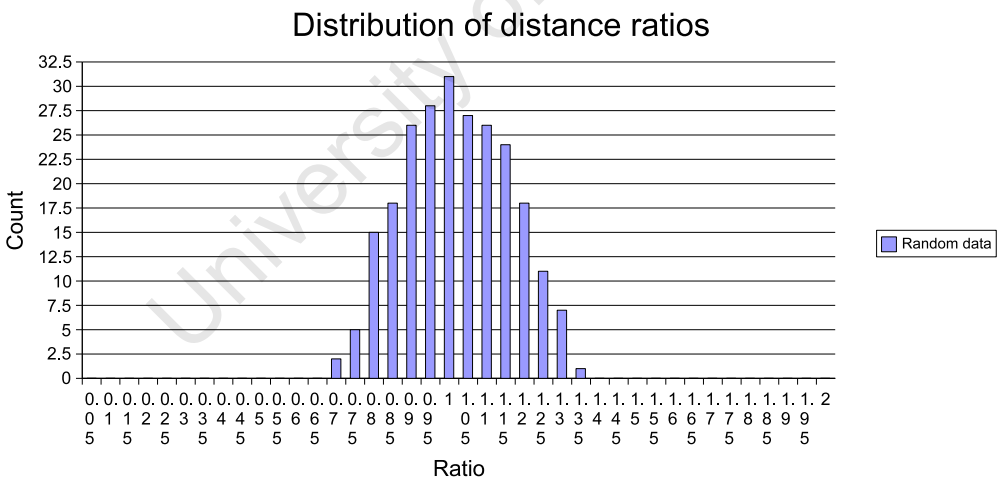
On the whole, all the evidence presented thus far indicates that there is a good degree of clustering in the data — in other words, that users on the whole have similar perceptions.

With regard to function approximation, the data was found to be insufficient for a sufficiently accurate function reconstruction. From this, it was concluded that either a single expert user or a group of users would be required to provide sufficient training data.

An interesting observation during the investigation was watching each user's interaction with the environment. The users showed a wide variety of comfort in navigating a virtual world from a



(a) Experimental data



(b) Random data

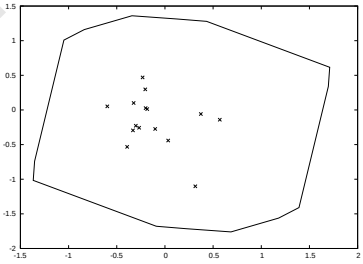
Figure 55: *Histograms for the data of the first investigation and for random data, showing the distribution of the ratio r_i from Equation 15 over the data sets.*

first-person perspective, with some exploring each environment fully and others only examining the world immediately around them. Although we did not explicitly ask users about their past experience in exploring virtual environments, we suggest that there is a strong correlation between past experience and the comfort in exploring a new environment. This observation implies that if further investigations were to also use virtual environments, that it might be better to show users single frames or images of the environment rather than allowing for exploration, so as to obtain consistent and objective results.

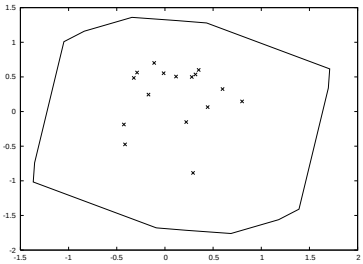
A.4 Summary

In summary, it was found in this this investigation that:

- Users prefer a categorical interface for selecting adjectival descriptors.
- Users have similar perceptions.
- A single user or group of expert users will be required to provide the core corpus of training data.
- To obtain unbiased data, care should be taken to keep the experience static across the spectrum of users.

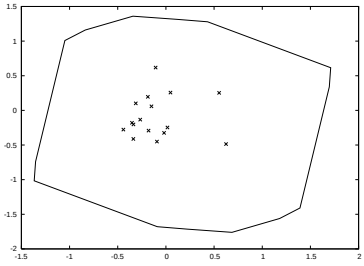


(a) Environment 1

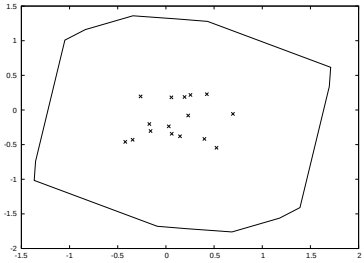


(b) Environment 2

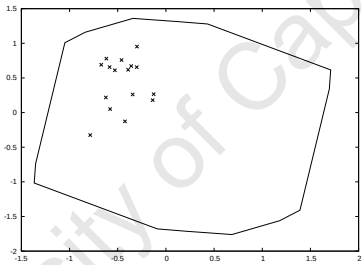
Figure 56: *2D visualisations of the data collected in the first investigation, environments 1 and 2.*



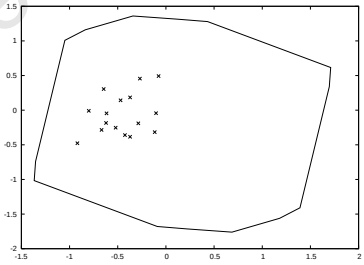
(a) Environment 3



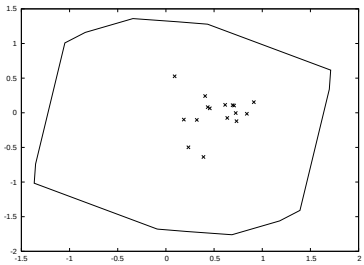
(b) Environment 4



(c) Environment 5

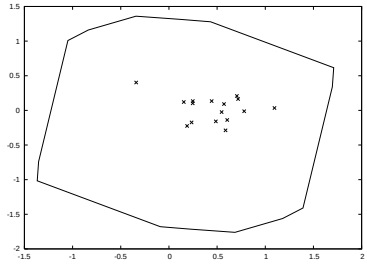


(d) Environment 6

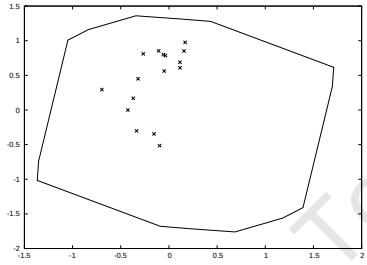


(e) Environment 7, 2D

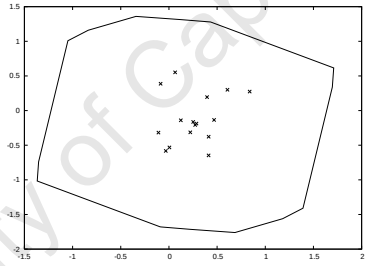
Figure 57: *2D visualisations of the data collected in the first investigation, environments 3-7.*



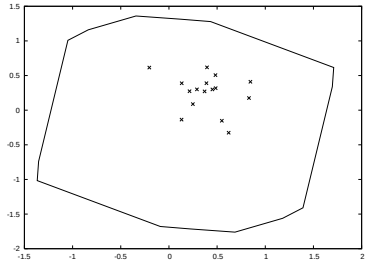
(a) Environment 8



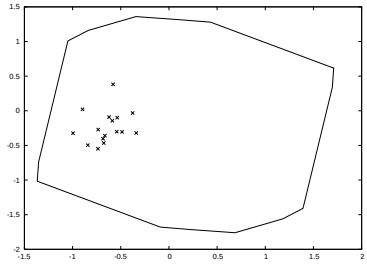
(b) Environment 9



(c) Environment 10

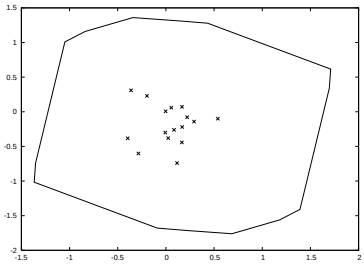


(d) Environment 11

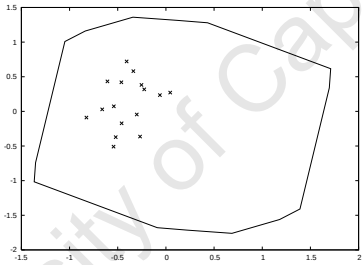


(e) Environment 12

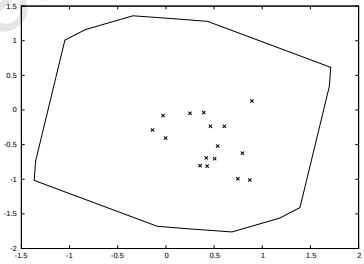
Figure 58: *2D visualisations of the data collected in the first investigation, environments 8-12.*



(a) Environment 13



(b) Environment 14, 2D



(c) Environment 15, 2D

Figure 59: *2D visualisations of the data collected in the first investigation, environments 13-15.*

Appendix B

Second investigation: large-scale data collection

Having established from our first investigation in Appendix A that users' opinions are largely similar, the main aim for the second investigation was to gather as much data as possible for better function approximation analysis and design.

B.1 Outline of investigation

In order to reach as many people as possible (and thus gather as much data as possible), the second investigation was posited as a survey on the internet. The same environment generator that was used in the first investigation was utilised to prepare 1000 different environments as a pre-process, randomly choosing the points in parameter space. By virtue of deciding to run this investigation on the web, and also in part through the observations in the first investigation that interactivity could lead to biased data, it was decided for each environment to display two static images — one showing the environment from an elevated distance view, and another closer shot giving a first-person perspective as might be observed during an interactive exploration. In an effort to reduce the burden of running successive investigations to collect different forms of data, users were randomly presented with one of three possible tasks:

1. **Standard scene description.** As with the first investigation, the user is shown an environment and asked to describe it using the descriptors provided. Taking on the users' preference for a categorical interface from the first investigation, seven categories were given for each descriptor, using the same seven descriptors from the first investigation. In addition, lighting and fog effects in the night environments could make it difficult for a user to accurately choose a suitable value for some of the descriptors, or in some rare cases the camera position and

angle chosen for the close image might also impair the user's ability to make a good judgement — for these reasons, an additional category was also provided for the user to indicate such circumstances.

2. **Adjective list.** Given a list of adjectives, the user is simply asked to indicate which ones apply to the environment shown.

3. **Adjective query.** The user is asked to describe the environment shown using whatever adjectives they feel are appropriate.

To maximise the amount of data acquired, users were able to repeat the process as many times as they wished — each iteration would give them a different environment or task to perform[1]. In addition, several environments were shown for each iteration through the process, where the number shown depended on which of the three tasks was given (as the time required to perform each of the tasks for a single environment differs quite substantially). Since the primary focus of this investigation was to collect data that could be used for function approximation, the probability of choosing the first task in the list above was weighted higher than that of choosing the other two tasks. The number of environments shown per iteration and probability of choosing a task are summarised in Table 20.

Task #	**Number of environments shown**	**Probability of selection**
Standard scene description	2	0.8
Adjective list	4	0.1
Adjective query	4	0.1

Table 20: *The number of environments shown and the probability of a task being chosen for each of the three tasks in our second investigation.*

B.2 Objectives

The primary purpose of the second investigation was to collect a large amount of data, for the purpose of testing the performance and accuracy of function approximation techniques. By collecting data from many people, an additional hope was that several of the environments would be described by more than one person, which would allow for the similarity in different participants' opinions to be analysed.

B.3 Results and discussion

In total, 306 people took part in the investigation, completing a total of 587 iterations. Table 21 shows how these iterations were distributed over the three tasks available, and Figure 60 shows the

[1] With an upper limit of having evaluated all 1000 environments for all three tasks.

distribution of the number of iterations completed by the participants.

Task	Iterations completed	Scenes per iteration	Effective data point count
Standard scene description	469	2	938
Adjective list	57	4	228
Adjective query	61	4	244

Table 21: *The number of iterations performed for each task in the second investigation, and the effective number of data points collected as a result.*

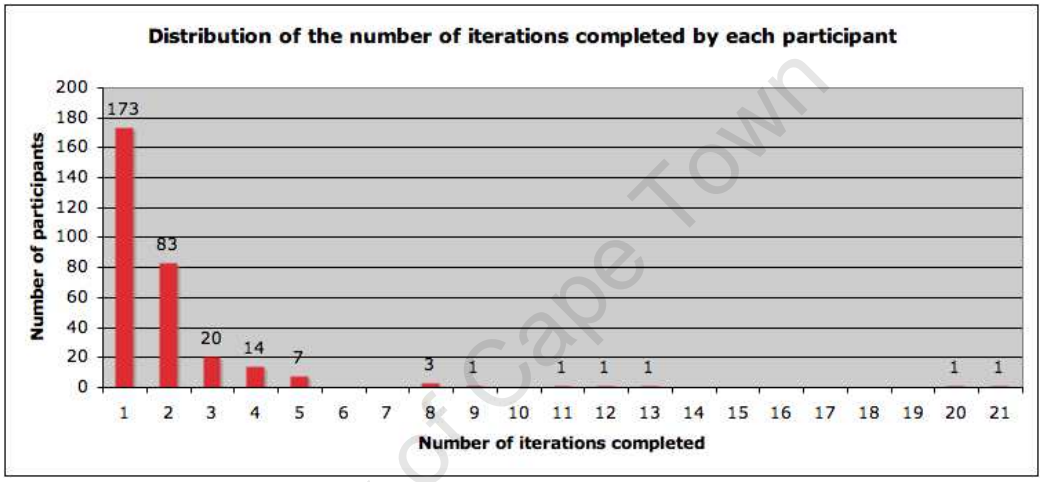


Figure 60: *A graph showing the distribution of the number of iterations completed by participants in the second investigation.*

To reinforce the conclusion from the first investigation that users have similar opinions, an analysis was done of the environments in this investigation that were described by more than one user. For each such environment, all pairwise combinations of users that described the environment were considered as individual data points, giving rise to a total of 451 data points over a total of 244 environments. Figure 61 shows box and whisker plots that illustrate the results of a statistical analysis of this data. These show that the means of the differences are all small — in all cases, the mean was 1, which indicates that on the whole users picked categories that were adjacent to each other.

Whilst the majority of the data, as indicated by the whiskers of the plot, falls within the lower half of the chart, one might hope for the whiskers to be lower and more constrained if users' perceptions were truly similar. One can also observe that the antonym-paired descriptors (such as wet/dry), in general exhibit better characteristics than the non-paired descriptors. This could be because the paired descriptors have their seven categories split over two adjectives, in effect meaning that there are fewer categories for each individual adjective.

For completeness, the distribution of the responses that users gave for each adjectival descriptor is included at the end of this chapter, in Figures 62, 63, 64, 65, 66, 67 and 68.

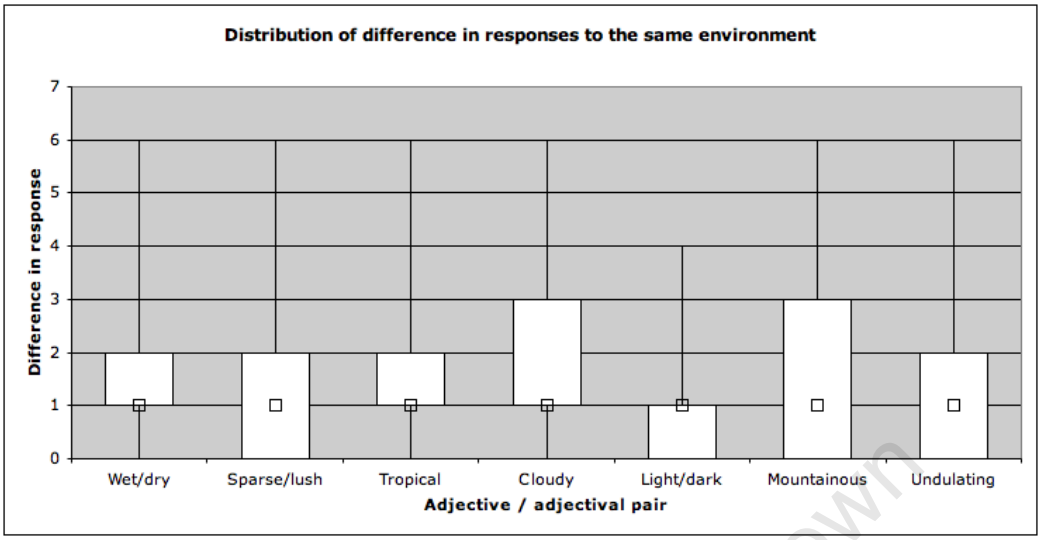


Figure 61: *Box and whisker plots showing how participants descriptions of common scenes differed.*

In Section 5.2, it was determined that RBFNs would be the most appropriate method of function approximation to meet the requirements of an adjectival interface. The data collected in this investigation then provided a good opportunity to test that the amount of data is sufficient for a suitably close function approximation. Recall from Section 4.2.2 that the *model selection criteria* (MSC) provides a metric for the degree to which the model correctly approximates the data, and that in particular the *leave-one-out* (LOO) estimator is particularly useful as the whole set of training data can be employed for both training and testing the model. The LOO estimator gives the average error of the model if one reserved a single data point for test data, and repeated this process for all n data points. Table 22 shows the values of this estimator for the RBFNs constructed for each adjectival descriptor.

Adjectival descriptor	$\sigma^2_{\mathbf{LOO}}$
Wet/dry	0.0074977
Sparse/lush	0.007058
Tropical	0.009258
Cloudy	0.013547
Light/dark	0.003465
Mountainous	0.010529
Undulating	0.008075

Table 22: *Values of the LOO estimator for RBFNs trained using the data from the second investigation.*

There are two important conclusions that can be drawn from the LOO estimator values:

1. **A sufficiently good approximation to the data was obtained.** Whilst the LOO values reported in Table 22 would be insufficient for a high-fidelity approximation, they are more than sufficient for the adjectival interface of this thesis.

2. **The LOO estimator values serve as a good indication of the similarity in user's opinions.** Interestingly, the larger LOO estimator values correspond to the adjectival descriptors that exhibited worse similarity scores. This indicates that the function approximation is more likely to perform better on data that reflects a similar user opinion, which reinforces the notion that an expert user or small group of expert users should be used to train the system.

B.4 Summary

In summary, it was found in this investigation that:

- The amount of data collected in the investigation provided for a sufficiently accurate function approximation.
- Users perceptions were again shown to be similar: more strongly so for the antonym-paired type of descriptors.

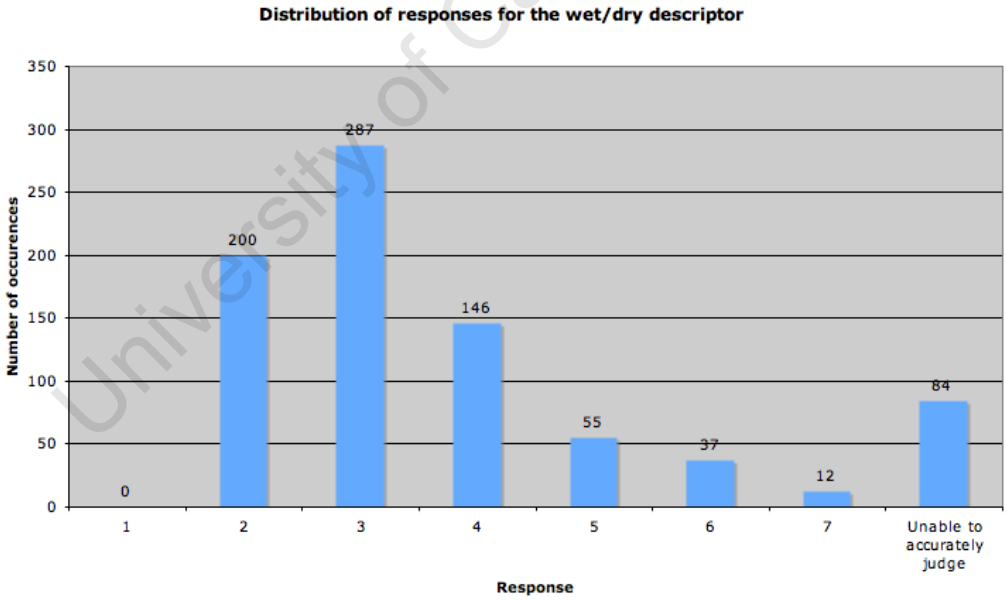


Figure 62: *Distribution of response for the wet/dry descriptor in the second investigation.*

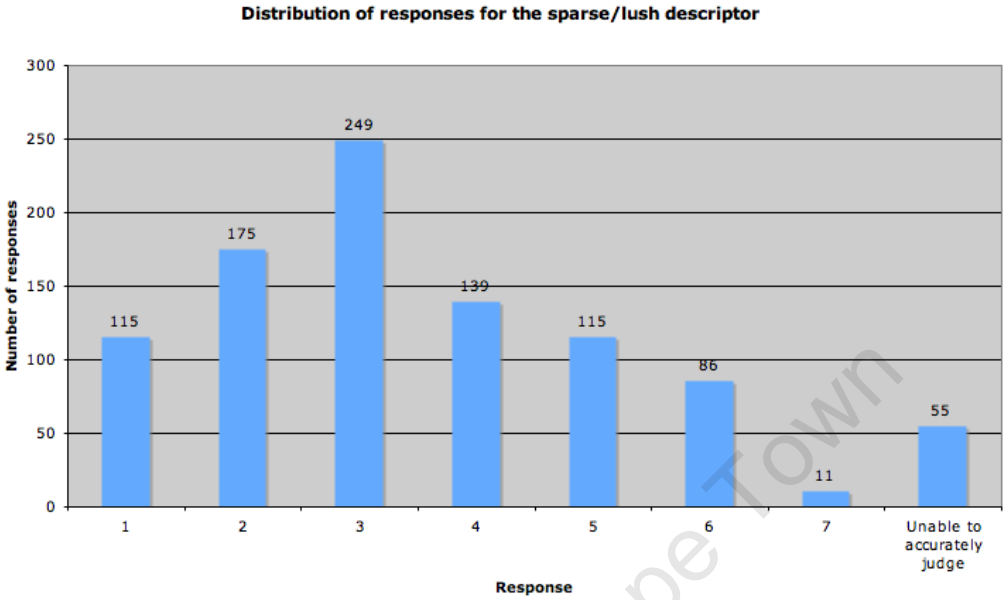


Figure 63: *Distribution of response for the sparse/lush descriptor in the second investigation.*

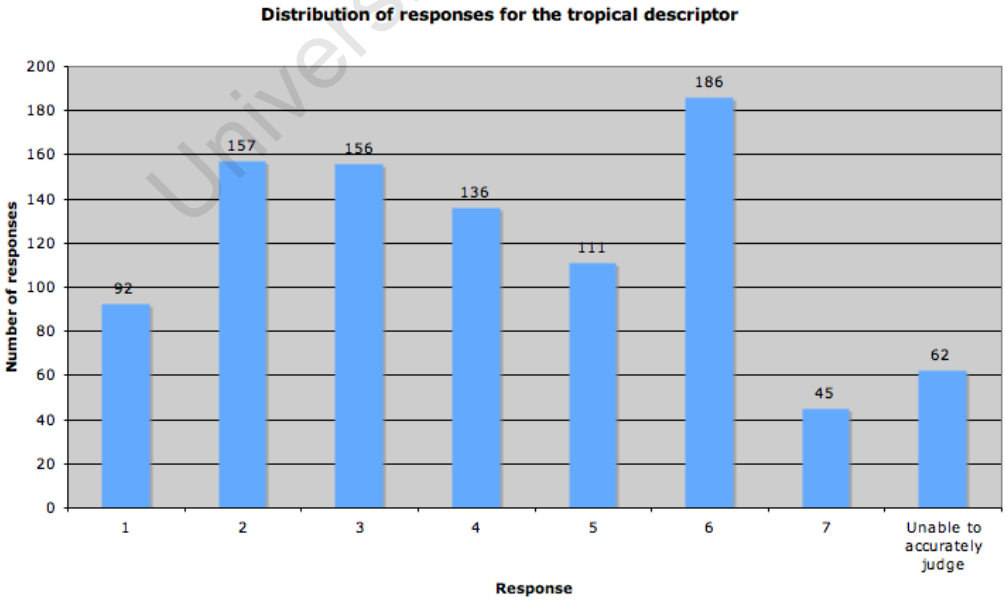


Figure 64: *Distribution of response for the tropical descriptor in the second investigation.*

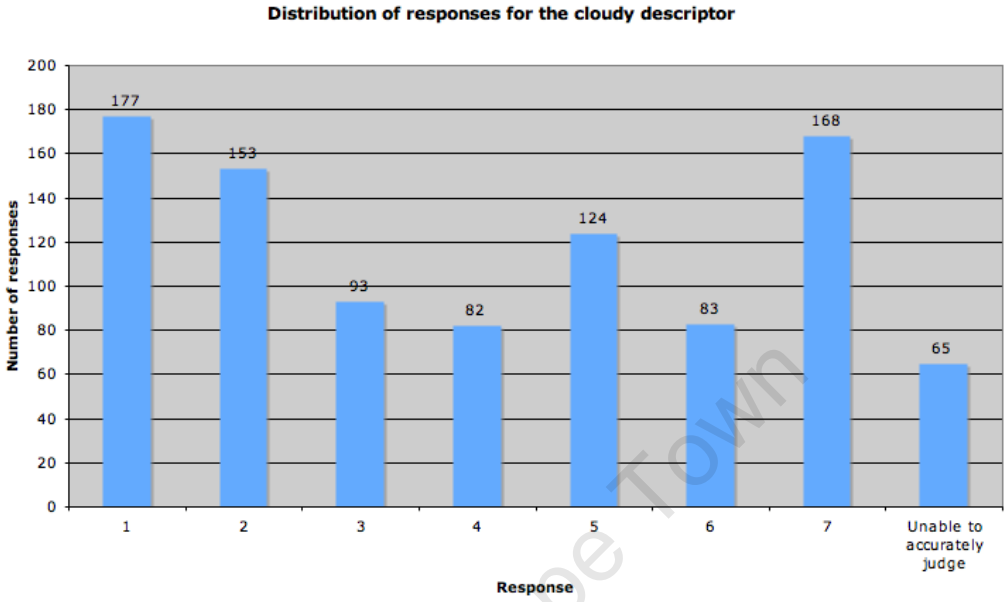


Figure 65: *Distribution of response for the cloudy descriptor in the second investigation.*

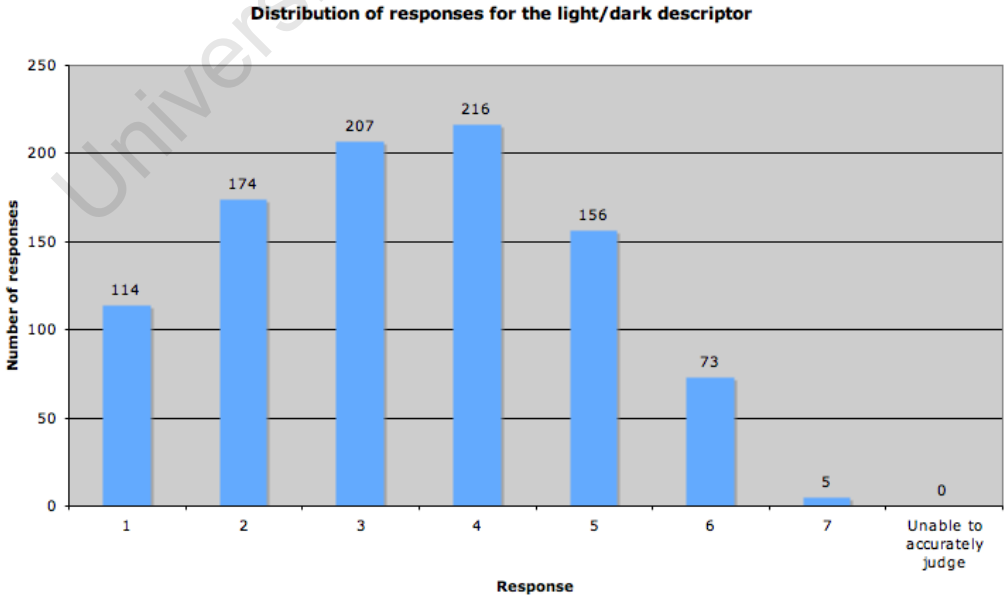


Figure 66: *Distribution of response for the light/dark descriptor in the second investigation.*

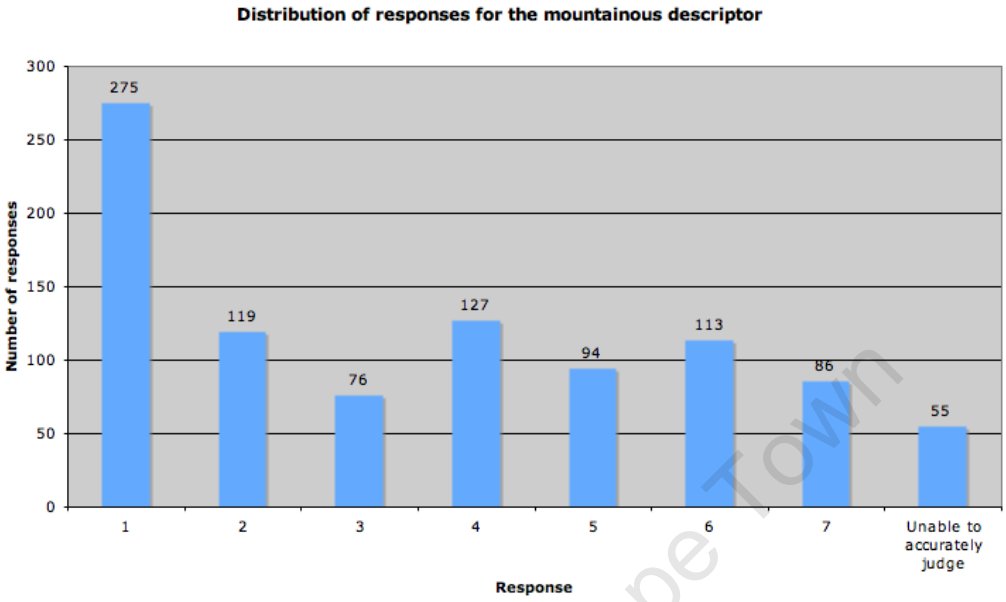


Figure 67: *Distribution of response for the mountainous descriptor in the second investigation.*

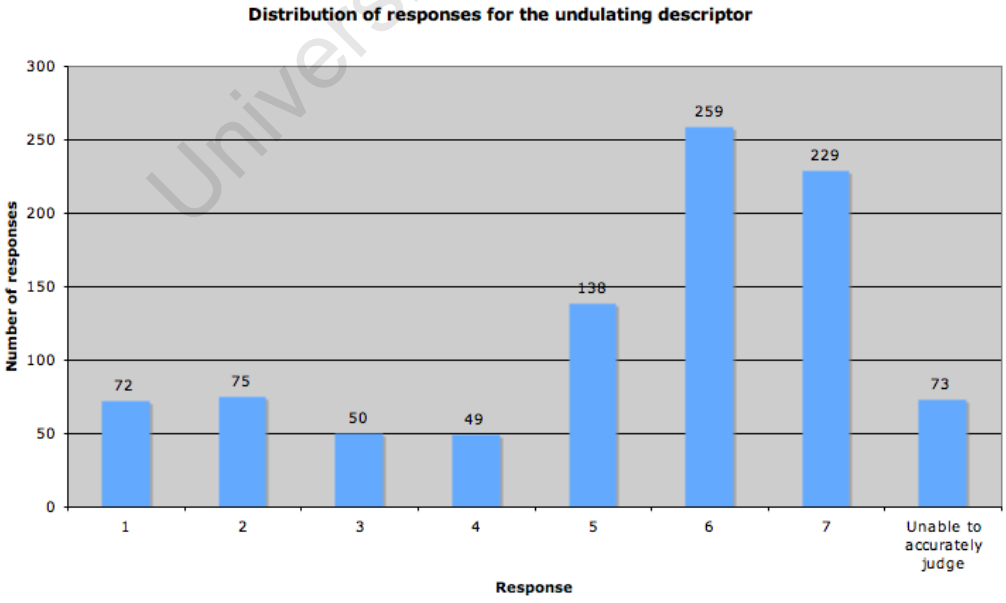


Figure 68: *Distribution of response for the undulating descriptor in the second investigation.*

Appendix C

Houdini procedural landscape system

As was briefly introduced in Chapter 6, Houdini [Side Effects Software Inc., 2008b] is a piece of software that is primarily used for the creation of 3D graphics content. It has been utilised in the production of several Hollywood movies, including Spider-Man 3 [Side Effects Software Inc., 2008c] and Beowulf [Side Effects Software Inc., 2008a], and offers many advanced controls for the generation of content. What characterises Houdini, though, is its procedural approach to modelling. Almost everything in Houdini is procedural, and represented by networks of procedural components that take inputs and produce outputs which can be consumed by other components. Figure 69 illustrates this process, showing part of a network for constructing a procedural landscape.

Each component is controlled via a set of procedural parameters, which can either be fixed at some value, evaluated as an expression[1] or exported as a global variable that can be adjusted by the user. This conveniently allows for some parameters to be set statically, and for others to be exported as parameters that can be controlled by an external user — the latter serving as procedural parameters for the overall model. In the system created for the experiment of Chapter 6, a total of 46 parameters were exposed in this way for the user to control — these are listed in Tables 23 and 24.

Geometric modelling is intrinsically complex, as the final rendering is not only dependent on the 3D geometry, but also on texture information, lighting and possibly animation. Houdini supports these various components through different network types, of which the following were pertinent to the procedural model created:

- **Geometry:** the geometry component provides a means for manipulating the physical construction of content, and allows for standard CSG operations as well as the ability to source

[1] Expressions here will typically reference animation environment variables, which allow the expression to evaluate differently over time.

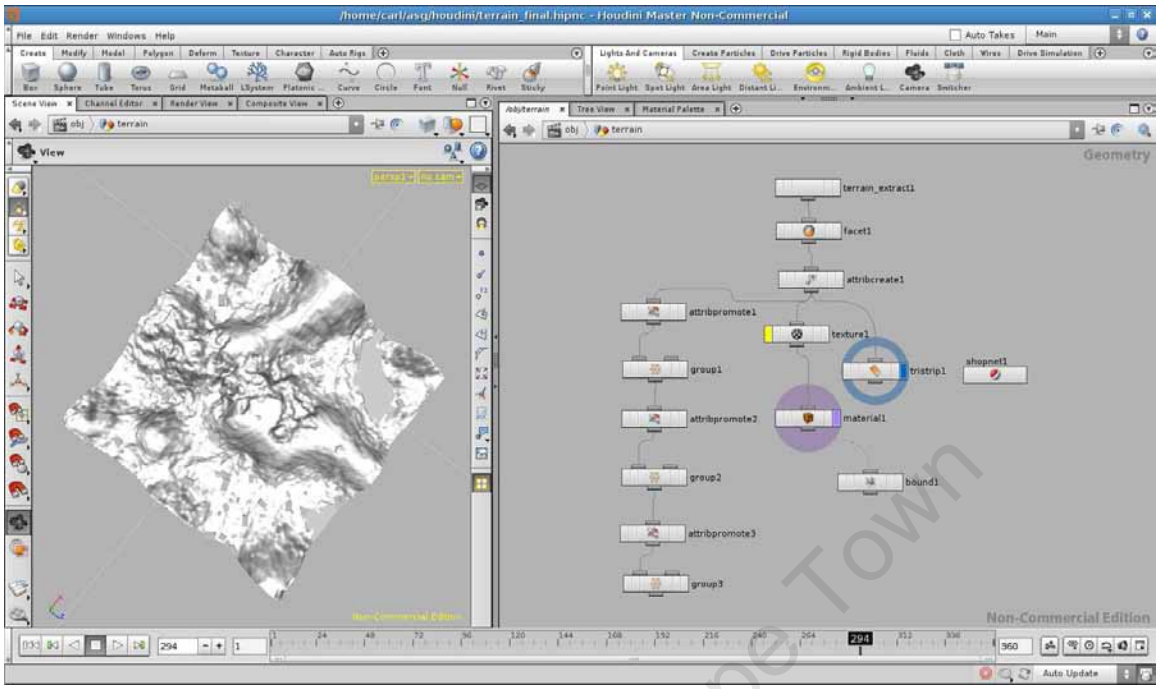


Figure 69: *An example of a procedural network in Houdini, showing how various components feed into each other.*

geomtery programatically. This latter feature is particularly useful, and allowed for the creation of terrain from a dynamically generated heightmap in the compositing component (see below).

- **Compositing:** the compositing component essentially allows for the procedural editing of 2D images, from simple operations that can be found in most image manipulation applications, to programatically controlling the output of each pixel.
- **Shaders:** procedural shaders allow for the dynamic colouring and lighting of objects, without the need to pre-specify any textures. In essence, every pixel that is rendered is passed through the shader to determine its final colour. As such, procedural shaders allow for the easy creation of dynamic content as well as infinite detail.

Some details of the procedural elements that were used are now discussed, with special reference to the different network types and how these elements interacted.

Parameter name	Minimum	Maximum	Default	Description
escarpment_top_extent	0	0.09	0	Extent of top escarpment
escarpment_top_amplitude	0	1.5	1	Amplitude of top escarpment
escarpment_top_freq_x	0	10	2	Frequency of noise for top escarpment
escarpment_top_offset	0	10	2	Offset of noise for top escarpment
escarpment_right_extent	0	0.09	0	Extent of left escarpment
escarpment_right_amplitude	0	1.5	1	Amplitude of left escarpment
escarpment_right_freq_x	0	10	2	Frequency of noise for left escarpment
escarpment_right_offset	0	10	2	Offset of noise for left escarpment
escarpment_left_extent	0	0.09	0	Extent of right escarpment
escarpment_left_amplitude	0	1.5	0	Amplitude of right escarpment
escarpment_left_freq_x	0	10	2	Frequency of noise for right escarpment
escarpment_left_offset	0	10	2	Offset of noise for right escarpment
central_highlands_centre_x	0	1	0.5	X centre for central highlands area
central_highlands_centre_y	0	1	0.5	Y centre for central highlands area
central_highlands_amplitude	0	5	1.4	Amplitude for central highlands area
central_highlands_offset	0	10	1.6	Noise offset for central highlands area
central_highlands_frequency	0	10	4.4	Noise frequency for central highlands area
central_highlands_comparison_value	0	0.1	0.03	Noise threshold for central highlands area
voronoi_noise_c1	-1	1	-1	Voronoi noise closest neighbour contribution
voronoi_noise_c2	0	1	1	Voronoi noise second closest neighbour contribution
voronoi_noise_scale	1	20	2.5	Voronoi noise scale
voronoi_noise_offset	0	10	0	Voronoi noise offset
perlin_noise_scale	1	20	2.5	Perlin noise scale
perlin_noise_offset	0	10	0	Perlin noise offset

Table 23: *List of procedural parameters used for third experiment (parameters 1 through 31 of 49).*

Parameter name	Minimum	Maximum	Default	Description
voronoi_perlin_blend_factor	0	1	0.66	Voronoi/Perlin noise blend factor
distortion_amplitude	0	1	0.4	Amplitude of heightmap distortion
smoothness	0	1	0	Terrain smoothness
max_height	5	25	10	Maximum terrain height
sea_level	0	0.7	0.1	Sea level
lake_max_pixels	0	1.0	0.3	Maximum coverage of terrain by lakes
lake_min_component_size	50	500	200	Minimum lake size
moisture	0	1	0.5	Average moisture level
beach_dilation	10	50	25	Maximum distance of beach from sea
beach_height	0	1	0.1	Maximum elevation of beach above sea
green_max_height	0	1	0.6	Maximum elevation for vegetation
green_max_gradient	0	1	0.2	Maximum gradient for vegetation
green_dirt_blend	-1	1	0.35	Blend factor between green and desert zones
palm_density	0	1	0.05	Palm tree density
green_zone_density	0	1	0.05	Density of vegetation in green zone
dirt_density env	0	1	0.05	Density of vegetation in desert zone
vegetation_scale	1	10	2	Vegetation scale
fir_proportion	0	5	1	Proportion of fir trees
asparagus_proportion	0	5	1	Proportion of asparagus plants
eastern_cottonwood_proportion	0	5	1	Proportion of eastern cottonwood trees
lombardy_poplar_proportion	0	5	1	Proportion of lombardy poplar trees
quaking_aspen_proportion	0	5	1	Proportion of quaking aspen trees
sassafras_proportion	0	5	1	Proportion of sassafras trees
cactus_proportion	0	5	1	Proportion of cactii
desert_bush_proportion	0	5	1	Proportion of desert bushes

Table 24: *List of procedural parameters used for third experiment (parameters 32 through 49 of 49).*

C.1 Terrain

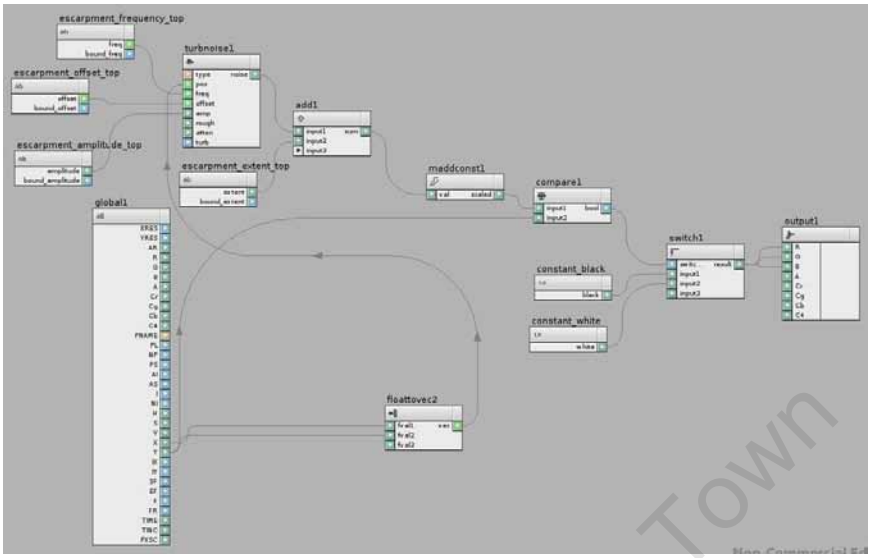
Terrain is an extremely important part of an outdoor landscape, as it provides the base that many of the other components rely on for suitable placement. The most common way of creating terrain is through the use of heightmaps, as discussed in Section 2.2.3. This mechanism is convenient, as it separates the problem of rendering the landscape from that of modelling it, and simplifies the modelling process to providing a two-dimensional greyscale image. In Houdini parlance, this is equivalent to creating a compositing component for generating the heightmap, and a geometry component which utilises that heightmap to generate geometry.

C.1.1 Modelling the terrain

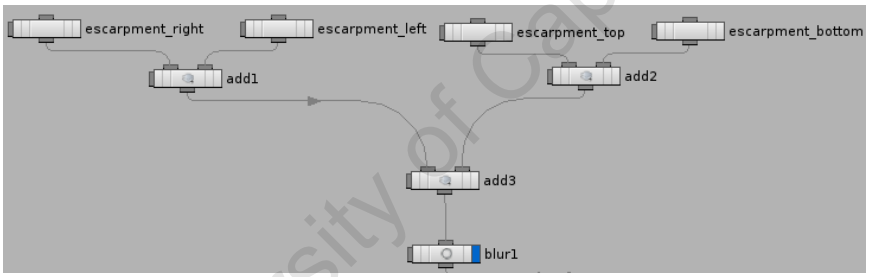
Modelling an "infinite" landscape is in itself a non-trivial task, and so this technique restricts the landscape to a square piece of land, where the borders are defined to either be high-lying escarpment or coastal regions. This lends itself to a multi-faceted terrain composition, in which the large-scale features and small-scale noise are modelled separately and then combined to give the overall terrain. The result of this modelling is a heightmap, which can then be used to create a heightfield representing the terrain.

Large-scale features

In addition to the border escarpment and coastal regions, a central mountainous area is also modelled — this combination of large-scale features allows for a diverse range of landscapes to be created. The creation of these regions involves the use of one-dimensional noise functions to generate a noisy curve defining the edge of the escarpment or central mountains, after which a Gaussian blur is applied to smoothly blend these large-scale features with the small-scale noise. See Figures 70 and 71 for examples of the escarpment and central mountains.



(a) The network for a single side of the escarpment. Four of these — one for each side — are used to produce an overall escarpment, as shown in Figure 70(b).

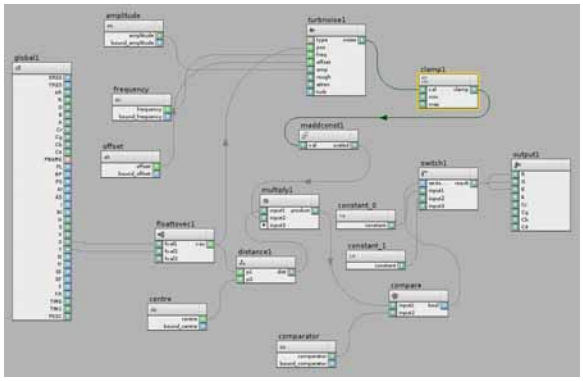


(b) The compositing network used to create the outer escarpment. Each of the four top-most nodes controls a single side, and contains a network similar to that shown in Figure 70(a) — there are slight differences to account for rotation and offset.

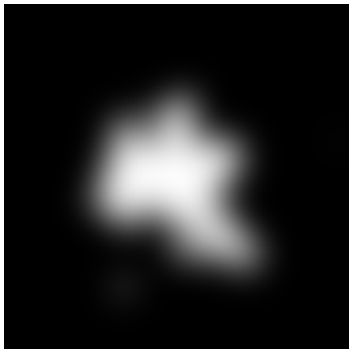
(c) An example of how the top part of the escarpment might look.

(d) An example of how the full escarpment might look (if escarpment were to be generated on all sides).

Figure 70: *The compositing networks used in creating an escarpment, and examples of the resulting heightmap produced.*



(a) The compositing network used to generate the central highlands area.

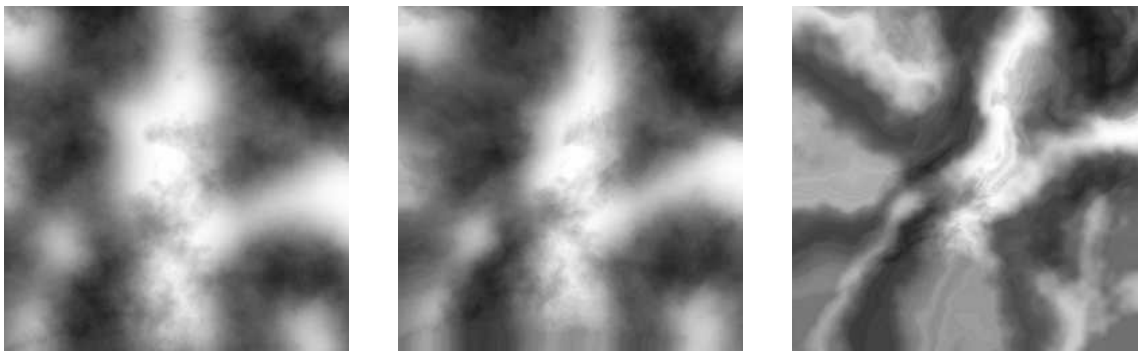


(b) An example of how the central highlands area might look.

Figure 71: *The compositing networks used in creating a central highlands area, and an example of the resulting heightmap produced.*

Small-scale noise and perturbation

Having generated larger landscape features, the next step is to generate convincing noisy terrain on a smaller scale. The approach adopted here very closely follows that of Olsen [2004], utilising a blend of Worley noise and Perlin noise. This is blended with the large-scale features, and the final result is then perturbed to eliminate any unnatural straight lines that have appeared as a result of the Worley noise. The perturbation also has the convenient effect of making the large-scale features more natural, by mixing them with the small-scale noise to give small mountainous outcrops and low-lying gulleys. Figure 72(a) illustrates a sample heightmap that combines large- and small- scale features, and Figures 72(b) and 72(c) illustrate the effects of varying distortion on this heightmap.



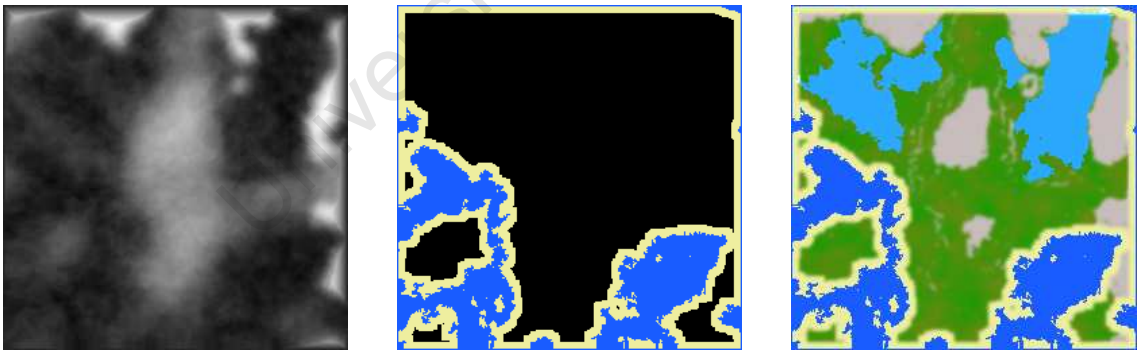
(a) (b) (c)

Figure 72: *An example of the final heightmap combining large- and small- features (a), and progressive distortion of this heightmap to simulate a more realistic feel (b-c).*

C.1.2 Shading the terrain

The composition of a terrain would be incomplete without appropriate colouring and shading — whilst greyscale heightmaps give a good impression of the physical layout of the terrain, suitable colouring hints at the vegetation and climate of the area. For simplicity, the model used here divides the terrain into four major regions — beach, desert, a green zone, and rocky highlands. The regions are allowed to merge into each other, giving varying degrees and mixtures of each. The determination of the regions is achieved using Houdini's compositing tools in the following ways:

- **Beach:** beach areas are typically located within close proximity of the sea. Once the initial heightmap has been created and the sea identified, choosing beach areas is a fairly simple matter of dilating the area covered by water. Figure 73(b) illustrates an example of this dilation process.
- **Rocky highlands:** high-lying areas, and areas wih steep gradient, are typically rocky in nature due to a lack of soil deposition. These are easily identified through appropriate compositing components.
- **Green zone and desert:** once the beach and rocky areas have been allocated, the remaining space is assigned to the green zone and desert areas. Perlin noise is used to blend continuously between the two areas, with the mean value biased to indicate the proclivity of the landscape for one or the other region.



(a) (b) (c)

Figure 73: *An example of assigning zones to the terrain. Figure 73(a) shows the heightmap being operated on, 73(b) shows how beach regions may be identified by dilation of the area covered by sea, and 73(c) shows a final allocation of zones. In the latter image, the colours dark blue, light blue, yellow, green, brown and grey, correspond to sea, lakes (see Section C.2), beach, green zone, desert and rocky highlands, respectively. Note that in some places the colours blend to indicate the merger of neighbouring regions.*

Figure 73(c) shows an example of a final allocation of regions over the heightmap. A simple, but unrealistic, solution to shading the terrain would be to apply this image as a texture. This approach is, however, problematic for several reasons:

1. **Poor resolution:** from a distance, the terrain might seem fairly realistic. As one moves closer toward the terrain, though, the static resolution of the texture causes the terrain to look increasingly like a patchwork of coloured squares, and less like a natural landscape. One could create a larger texture, but this requires an enormous amount of storage — and any texture, no matter how big, will at some scale be too coarse and cause an unnatural appearance.

2. **Vertical texture distortion:** because terrain is three-dimensional, areas on the map with steep gradients will cause the texture to warp and distort unnaturally.

To avoid the pitfalls of texture mapping, procedural shaders are instead used to colour the rendered terrain. For each region, a small exemplar texture is first chosen that captures the look of the region and the three-dimensional coordinates of the point being shaded are used as a lookup into this texture. In order to avoid the repetition that would result if the landscape were flat, the coordinates are also perturbed using Perlin noise before indexing the texture. Since certain parts of the terrain are located at the merger of two or more regions, a texture lookup is done for each region and these are blended together appropriately. Finally, lighting calculations are applied to simulate diffuse light, based on the position of the sun. Figure 74 shows an example of the final rendered terrain, using the region allocation of Figure 73(c).

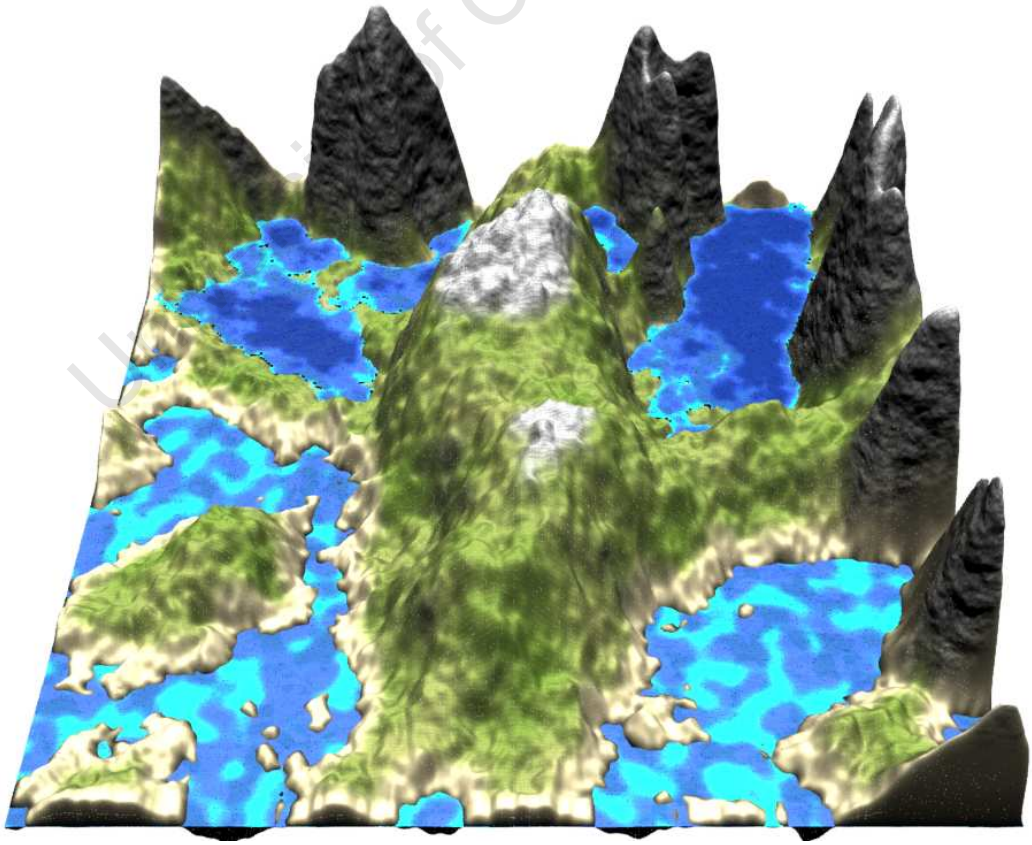


Figure 74: *An example of shading of the terrain as applied to the region allocation shown in Figure 73(c).*

C.2 Rivers and lakes

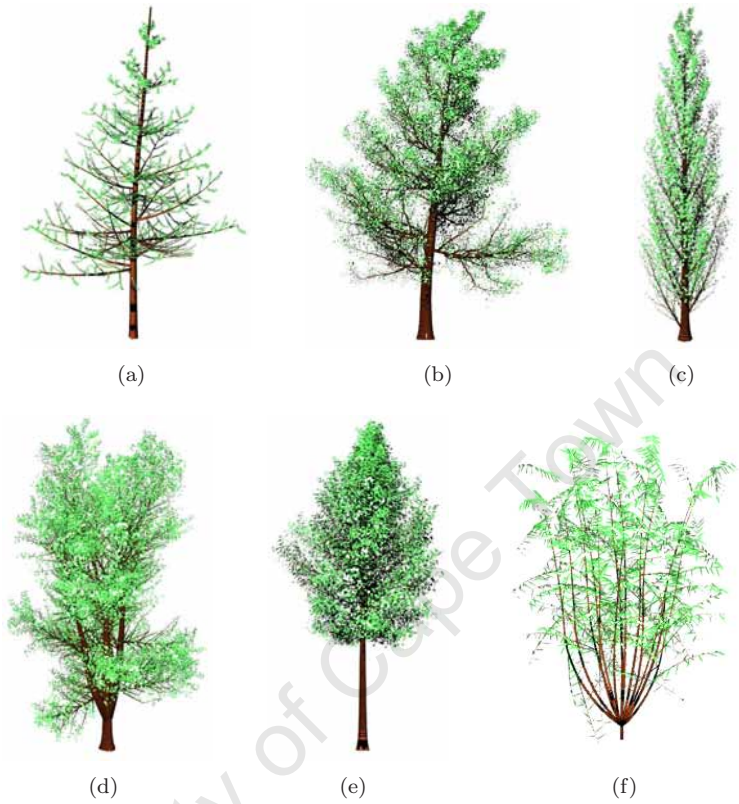
Simply having landscape would be somewhat static and boring — for a more natural setting, rivers and lakes are also added to the environment. Once a heightmap has been generated, the height and gradient data can be used to determine where water might pool — giving lakes — and escape to reach other lakes, or eventually reach the sea — giving rivers. This is done via a custom-built plugin that can be integrated into Houdini via a modular API. Figure 73(c) shows the location of two lakes for an example heightmap, which can also be seen in the accompaning rendering in Figure 74.

C.3 Trees and scrubs

A truly procedural environment would make use of unique trees and plants, each of which is procedurally generated. The costs of doing this are, however, prohibitively large, and so for this landscape, several trees and plants are pre-created and then instanced in appropriate vegetation regions with varying size and rotation attributes to avoid monotony. Specifically, two forms of vegetation are available for the desert region, six for the green zone, and one for beaches. Figures 75, 76 and 77 show the various trees and plants that are used, which were all created using Arbaro [Diestel, 2003].

(a) (b)

Figure 75: *The instances of vegetation used for desert regions.*



(a) (b) (c)

(d) (e) (f)

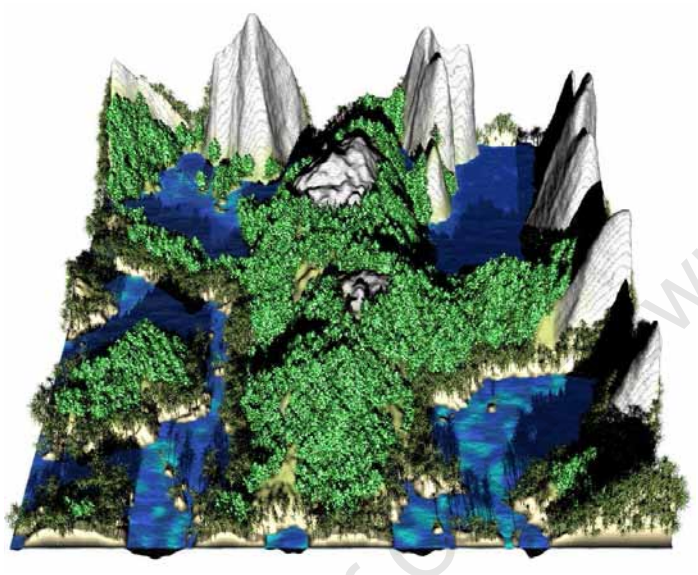
Figure 76: *The instances of vegetation used for green zone regions.*

(a)

Figure 77: *The instance of vegetation used for beach regions*

C.4 Putting everything together

Figures 78(a) shows a complete landscape that comprises all the elements discussed, and Figure 78(b) shows a closer view from a different angle.



(a)

(b)

Figure 78: *An example of the fully rendered landscape (a), and a closer rendering from a different angle (b).*

Bibliography

Anders Adamson and Marc Alexa. Anisotropic point set surfaces. In *Afrigaph '06: Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, pages 7–13, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-288-7.

Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Point set surfaces. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 21–28, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7803-7200-X.

Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, 2003. ISSN 1077-2626.

Daniel G. Aliaga, Paul A. Rosen, and Daniel R. Bekins. Style grammars for interactive visualization of architecture. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):786–797, 2007. ISSN 1077-2626.

David M. Allen. The relationship between variable selection and data agumentation and a method for prediction. *Technometrics*, 16(1):125–127, feb 1974. ISSN 0040-1706.

Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust. In *SMA '01: Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 249–266, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-366-9.

Isaac Amidror. Scattered data interpolation methods for electronic imaging systems: a survey. *Journal of Electronic Imaging*, 11(2):157–76, 2002. doi: 10.1117/1.1455013. Lab. de Syst. Peripheriques, Ecole Polytech. Fed. de Lausanne, Switzerland.

Katy Appleton and Andrew Lovett. *Display methods for real-time landscape models: an initial comparison*, pages 246–253. Wichmann, Heidelberg, 2005.

Ivo Babuška, Uday Banerjee, and John E. Osborn. Survey of meshless and generalized finite element methods: A unified approach. *Acta Numerica*, 12:1–125, 2003.

Norman I. Badler, Rama Bindiganavale, Jan Allbeck, William Schuler, Liwei Zhao, and Martha Palmer. Parameterized action representation for virtual human agents. pages 256–284, 2000.

Mark R. Baker and Rajendra B. Patil. Universal approximation theorem for interval neural networks. *Reliable Computing*, 4(3):235–239, August 1998.

Shumeet Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning,. Technical Report CMU-CS-94-163, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1994.

Kobus Barnard and David Forsyth. Learning the semantics of words and pictures. *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, 2:408–415 vol.2, 2001.

William V. Baxter and Ken ichi Anjyo. Latent doodle space. *Computer Graphics Forum*, 25(3): 477–485, September 2006.

Edward Bedwell and David Ebert. Artificial evolution of algebraic surfaces. In *Proceedings of Implicit Surfaces '99*, September 1999.

Richard E. Bellman. *Adaptive Control Processes*. Princeton University Press, 1961.

Bedrich Benes and Rafael Forsbach. Visual simulation of hydraulic erosion. In *WSCG*, pages 79–94, 2002.

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975. ISSN 0001-0782.

James F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. Graph.*, 1(3):235–256, 1982. ISSN 0730-0301.

Jean-Daniel Boissonnat and Frédéric Cazals. Smooth surface reconstruction via natural neighbour interpolation of distance functions. In *SCG '00: Proceedings of the sixteenth annual symposium on Computational geometry*, pages 223–232, New York, NY, USA, 2000. ACM Press. ISBN 1-58113-224-7.

László Böszörményi, Jürg Gutknecht, and Gustav Pomberger, editors. *The School of Niklaus Wirth, "The Art of Simplicity"*. dpunkt.verlag/Copublication with Morgan-Kaufmann, 2000. ISBN 3-932588-85-1.

Antoine Bouthors and Fabrice Neyret. Modeling clouds shape. In E. Galin M. Alexa, editor, *Eurographics (short papers)*, august 2004. URL `http://www-evasion.imag.fr/Publications/2004/BN04`.

Doug A. Bowman and Larry F. Hodges. Formalizing the design, evaluation, and application of interaction techniques for immersive virtual environments. *Journal of Visual Languages and Computing*, 10(1):37–53, 1999.

Adrian Bowyer. Computing dirichlet tessellations. *The Computer Journal*, 2(24):162–166, 1981.

Matthew Brand and Aaron Hertzmann. Style machines. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 183–192, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. ISBN 1-58113-208-5.

Leo Breiman, Jerome Friedman, Charles J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Chapman & Hall/CRC, January 1984. ISBN 0412048418.

David Broomhead and David Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

David C. Brown and Balakrishnan Chandrasekaran. Design considerations for picture production in a natural language graphics system. *SIGGRAPH Comput. Graph.*, 15(2):174–207, 1981. ISSN 0097-8930.

Sarah Brown, Ilda Ladeira, Cara Winterbottom, and Edwin Blake. An investigation on the effects of mediation in a storytelling virtual environment. Technical Report CS02-08-00, University of Cape Town, 2002.

Jonathan C. Carr, W. Richard Fright, and Richard K. Beatson. Surface interpolation with radial basis functions for medical imaging. *IEEE Transactions on Medical Imaging*, 16(1):96–107, February 1997.

Jonathan C. Carr, Richard K. Beatson, Jon B. Cherrie, Toby J. Mitchell, W. Richard Fright, Bruce C. McCallum, and Tim R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-374-X.

N.H. Christ, R. Friedberg, and T.D. Lee. Weights of links and plaquettes in a random lattice. *Nuclear Physics B*, 210(3):337–346, 1982.

R. Clough and J. Tocher. Finite element stiffness matrices for analysis of plates in blending. In *In Proceedings of Conference on Matrix Methods in Structural Analysis*, 1965.

Matthew Conway, Steve Audia, Tommy Burnette, Dennis Cosgrove, and Kevin Christiansen. Alice: lessons learned from building a 3d system for novices. In *CHI*, pages 486–493, 2000.

Robert L. Cook and Tony Derose. Wavelet noise. *ACM Trans. Graph.*, 24 (3):803–811, 2005. ISSN 0730-0301. doi: 10.1145/1073204.1073264. URL `http://dx.doi.org/10.1145/1073204.1073264`.

Trevor F. Cox and Michael A. Cox. *Multidimensional Scaling, Second Edition*. Chapman & Hall/CRC, September 2000. ISBN 1584880945.

Bob Coyne and Richard Sproat. Wordseye: an automatic text-to-scene conversion system. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 487–496, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-374-X.

Charles Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, 1859.

Jules Davidoff, Ian Davies, and Debi Roberson. Colour categories in a stone-age tribe. *Nature*, 398(6724):203–204, March 1999. ISSN 0028-0836. doi: 10.1038/18335. URL `http://dx.doi.org/10.1038/18335`.

Richard Dawkins. *The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe without Design*. W.W. Norton & Co., September 1996. ISBN 0393315703. URL `http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0393315703`.

Christopher de Kadt. Digital reconstruction of district six architecture from archival photographs. Master's thesis, University of Cape Town, 2007.

Matthew de Villiers and Neilan Naicker. A sketching interface for procedural city generation. Technical Report CS06-04-00, Department of Computer Science, University of Cape Town, November 2006.

Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *SIGGRAPH 96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 11–20, New York, NY, USA, 1996. ACM Press. ISBN 0-89791-746-4.

Boris Delaunay. Sur la sphère vide. *Izvestia Akademii Nauk SSSR*, 7:793–800, 1934.

Oliver Deussen, Pat Hanrahan, Bernd Lintermann, Radomr Měch, Matt Pharr, and Przemyslaw Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. In *SIGGRAPH 98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 275–286, New York, NY, USA, 1998. ACM Press. ISBN 0-89791-999-8.

Wolfram Diestel. Arbaro — tree generation for povray. `http://arbaro.sourceforge.net`, 2003.

Gustav Lejeune Dirichlet. Über die reduktion der positiven quadratischen formen mit drei unbestimmten ganzen zahlen. *Journal für die Reine und Angewandte Mathematik*, 40:209–227, 1850.

Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Willey & Sons, New Yotk, 1973.

George H. Dunteman. *Principal Components Analysis (Quantitative Applications in the Social Sciences)*. Sage Publications, 1989.

Theirry Dutoit, Vincent Pagel, Nicolas Pierret, Francois Bataille, and Olivia van der Vrecken. The MBROLA project: Towards a set of high quality speech synthesizers free of use for non commercial purposes. In *Proc. ICSLP '96*, volume 3, pages 1393–1396, Philadelphia, PA, 1996.

David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing and modeling: a procedural approach*. Academic Press Professional, Inc., 1994. ISBN 0-12-228760-6.

Karlheinz Essl. Lexikon-sonate. an interactive realtime composition for computer-controlled piano. In *II Brazilian Symposium on Computer Music*. Canela, 1995.

Kenneth Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. John Wiley & Sons, 2nd edition edition, September 2003.

Tom G. Farr, Paul A. Rosen, Edward Caro, Robert Crippen, Riley Duren, Scott Hensley, Michael Kobrick, Mimi Paller, Ernesto Rodriguez, Ladislav Roth, David Seal, Scott Shaffer, Joanne Shimada, Jeffrey Umland, Marian Werner andMichael Oskin, Douglas Burbank, and Douglas Alsdorf. The shuttle radar topography mission. *Reviews of Geophysics*, 45, 2007.

Christiane Fellbaum. *WordNet – An Electronic Lexical Database*. MIT Press, 1998.

Clive Fencott. Towards a design methodology for virtual environments, 1999.

Shachar Fleishman, Daniel Cohen-Or, and Cláudio T. Silva. Robust moving least-squares fitting with sharp features. *ACM Trans. Graph.*, 24(3):544–552, 2005. ISSN 0730-0301.

Alain Fournier, Don Fussell, and Loren Carpenter. Computer rendering of stochastic models. *Commun. ACM*, 25(6):371–384, 1982. ISSN 0001-0782.

Manuel Gamito. Procedural landscapes with overhangs. presented at the 10th Portuguese Computer Graphics Meeting and available at `http://www.dcs.shef.ac.uk/~mag/papers/epcg10.pdf`, 2001.

Sucharita Ghosh. *Projection pursuit type applications of multivariate empirical characteristic functions*. Mimeo series. Dept. of Statistics, University of North Carolina at Chapel Hill, 1989.

Alexander Goldberg, Matthias Zwicker, and Frédo Durand. Anisotropic noise. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–8, New York, NY, USA, 2008. ACM. doi: http://doi.acm.org/10.1145/1399504.1360653.

David Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison Wesley, 1989.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)*. The Johns Hopkins University Press, October 1996a. ISBN 0801854148.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd ed. edition, 1996b.

Gene H. Golub, Michael Heath, and Grace Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223, may 1979. ISSN 0040-1706.

Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popović. Style-based inverse kinematics. In *Proceedings of SIGGRAPH 2004*. ACM Press / ACM SIGGRAPH, 2004.

Stefan Gustavson. Simplex noise demystified. `http://staffwww.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf`, 2005.

John M. Hammersley and David C. Handscomb. *Monte Carlo Methods*. Methuen, London, 1964.

Mark J. Harris. Real-time cloud simulation and rendering. Technical Report TR03-040, University of North Carolina, 2003.

Mark J. Harris and Anselmo Lastra. Real-time cloud rendering. In A. Chalmers and T.-M. Rhyne, editors, *EG 2001 Proceedings*, volume 20(3), pages 76–84. Blackwell Publishing, 2001.

Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall, 1994.

Robert Hecht-Nielsen. Kolmogorov's mapping neural network existence theorem. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 11–13. IEEE Press, 1987.

Carl Helstrom. *Probability and Stochastic Processes for Engineers*. Macmillan, New York, 1984. ISBN 0023535601.

Øyvind Hjelle and Morten Dæhlen. Multilevel least squares approximation of scattered data over binary triangulations. *Computing and Visualization in Science*, 8(2):83–91, April 2005.

Douglas R. Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books, Inc., New York, NY, USA, 1979. ISBN 0465026850.

John H. Holland. *Hidden order: how adaptation builds complexity*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1995. ISBN 0-201-40793-0.

John H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

Eugene Hsu, Kari Pulli, and Jovan Popović. Style translation for human motion. *ACM Transactions on Graphics*, 24(3):1082–1089, August 2005.

Carl Hultquist, James Gain, and David Cairns. Affective scene generation. In *Afrigaph '06: Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, pages 59–63, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-288-7.

Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: a sketching interface for 3d freeform design. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 409–416, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-48560-5.

Takashi Ijiri, Shigeru Owada, Makoto Okabe, and Takeo Igarashi. Floral diagrams and inflorescences: interactive flower modeling using botanical structural constraints. *ACM Transactions on Graphics*, 24(3):720–726, 2005. ISSN 0730-0301.

ILO. Global employment trends brief. Report by the International Labour Organization (ILO), January 2007.

Shari L. Jackson, Joseph Krajcik, and Elliot Soloway. The design of guided learner-adaptable scaffolding in interactive learning environments. In *CHI '98: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 187–194, New York, NY, USA, 1998. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-30987-4.

Richard A. Johnson and Dean W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, 5th edition edition, 2002.

Dhiraj Joshi, James Z. Wang, and Jia Li. The story picturing engine—a system for automatic text illustration. *ACM Trans. Multimedia Comput. Commun. Appl.*, 2(1):68–89, 2006. ISSN 1551-6857.

Kenneth M Kahn. *Creation of computer animation from story descriptions*. PhD thesis, Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science, 1979.

Olga A. Karpenko and John F. Hughes. Smoothsketch: 3d free-form shapes from complex sketches: Copyright restrictions prevent acm from providing the full text for this work. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, page 13, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-364-6.

H. Kazmierczak and K. Steinbuch. Adaptive systems in pattern recognition. *IEEE Transactions on Electronic Computers*, 12:822–835, 1963.

Alex D. Kelley, Michael C. Malin, and Gregory M. Nielson. Terrain simulation using a model of stream erosion. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 263–268, New York, NY, USA, 1988. ACM. ISBN 0-89791-275-6.

C. Tim Kelley. *Solving nonlinear equations with Newton's method*. Fundamentals of Algorithms. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2003.

James Kennedy and Russell C. Eberhart. *Swarm intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001. ISBN 1-55860-595-9.

Gregor Kjellström. On the efficiency of gaussian adaptation. *J. Optim. Theory Appl.*, 71(3):589–597, 1991. ISSN 0022-3239.

Stephen C. Kleene. Representation of events in nerve nets and finite automata. In C. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–42. Princeton University Press, Princeton, N.J., 1956.

Peter Lancaster and Kes Salkauskas. Surfaces generated by moving least squares methods. *Mathematics of Computation*, 87:141–158, 1981.

Neil D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

Charles L. Lawson and Richard J. Hanson. *Solving Least Squares Problems*. Prentice-Hall, 1974.

Henri Lebesgue. *Intégrale, longueur, aire*. PhD thesis, University of Paris, 1902.

David Levin. The approximation power of moving least-squares. *Mathematics of Computation*, 67 (224):1517–1531, 1998.

Maxime Lhuillier. A quasi-dense approach to surface reconstruction from uncalibrated images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(3):418–433, 2005. ISSN 0162-8828. Senior Member-Long Quan.

Aristid Lindenmayer. Mathematical models for cellular interaction in development, parts i and ii. *Journal of Theoretical Biology*, 18(3):280–315, 1968.

C. Karen Liu, Aaron Hertzmann, and Zoran Popović. Learning physics-based motion style with nonlinear inverse optimization. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 1071–1081, New York, NY, USA, 2005. ACM.

William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, New York, NY, USA, 1987. ACM Press. ISBN 0-89791-227-6.

James MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics and Probability*, 1:281–296, 1967.

Benoit B. Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freedman and Co., New York, 1983.

Martin Marinov and Leif Kobbelt. Optimization techniques for approximation with subdivision surfaces. In *SM '04: Proceedings of the ninth ACM symposium on Solid modeling and applications*, pages 113–122, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association. ISBN 3-905673-55-X.

Joe Marks, Brad Andalman, Paul A. Beardsley, William Freeman, Sarah Gibson, Jessica Hodgins, Tom Kang, Brian Mirtich, Hanspeter Pfister, Wheeler Ruml, Kathy Ryall, Josh Seims, and Stuart Shieber. Design galleries: a general approach to setting parameters for computer graphics and animation. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 389–400, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co. ISBN 0-89791-896-7.

Donald H. McBurney and Theresa L. White. *Research Methods*. Wadsworth Publishing, 7 edition, 2006. ISBN 0-495-09208-8.

Warren S. McCulloch and Walter H. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

Radomír Měch and Przemyslaw Prusinkiewicz. Visual models of plants interacting with their environment. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics*

and interactive techniques, pages 397–410, New York, NY, USA, 1996. ACM Press. ISBN 0-89791-746-4.

Bruce Merry, Gianni Giacchetta, Bruce Thwaits, and James Gain. Genetic selection of parametric scenes. Technical Report CS03-12-00, Department of Computer Science, University of Cape Town, 2003.

Gavin S P Miller. The definition and rendering of terrain maps. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 39–48, New York, NY, USA, 1986. ACM. ISBN 0-89791-196-2.

Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, Massachusetts, 1969.

Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. *ACM Trans. Graph.*, 25(3):614–623, 2006. ISSN 0730-0301.

Pascal Müller, Gang Zeng, Peter Wonka, and Luc Van Gool. Image-based procedural modeling of facades. *ACM Trans. Graph.*, 26(3):85, 2007. ISSN 0730-0301.

F. Kenton Musgrave, Craig E. Kolb, and Robert S. Mace. The synthesis and rendering of eroded fractal terrains. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 41–50, New York, NY, USA, 1989. ACM. ISBN 0-201-50434-0.

Isaac Newton. Methodus fluxionum et serierum infinitarum, 1664–1671.

Shaun Nirenstein, Edwin Blake, and James Gain. Exact from-region visibility culling. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 191–202, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association. ISBN 1-58113-534-3.

Robert L. Ogniewicz and Olaf Kübler. Hierarchic Voronoi skeletons. *Pattern Recognition*, 28(3): 343–359, 1995.

Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. 3d scattered data approximation with adaptive compactly supported radial basis functions. In Franca Giannini and Alexander Pasko, editors, *Shape Modeling International 2004 (SMI 2004)*, pages 31–39, Genova, Italy, June 2004. IEEE. ISBN 0-7695-2075-8.

Makoto Okabe and Takeo Igarashi. 3d modeling of trees from freehand sketches. In *Proceedings of the SIGGRAPH 2003 conference on Sketches & applications*, pages 1–1, New York, NY, USA, 2003. ACM Press.

Jacob Olsen. Realtime procedural terrain generation. Oddlabs, `http://oddlabs.com/download/terrain_generation.pdf`, 2004.

Mark J. L. Orr. Introduction to radial basis function networks, April 1996.

Mark J. L. Orr. Recent advances in radial basis function networks. Technical report, Institute for Adaptive and Neural Computation, Division of Informatics, Edinburgh University, June 1999.

Pierre-Yves Oudeyer. The production and recognition of emotions in speech: features and algorithms. *International Journal of Human Computer Interaction*, 59(1-2):157–183, 2003. special issue on Affective Computing.

Biswanath Panda, Mirek Riedewald, Stephen B. Pope, Johannes Gehrke, and L. Paul Chew. Indexing for function approximation. In *VLDB'2006: Proceedings of the 32nd international conference on Very large data bases*, pages 523–534. VLDB Endowment, 2006.

Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308. ACM Press, 2001. ISBN 1-58113-374-X.

Ken Perlin. An image synthesizer. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 287–296, New York, NY, USA, 1985. ACM Press. ISBN 0-89791-166-0.

Ken Perlin. Improving noise. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 681–682, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-521-1.

Ken Perlin. Noise hardware. In Olano M., editor, *Real-Time Shading SIGGRAPH Course Notes*. 2001.

Tomaso Poggio and Federico Girosi. A theory of networks for approximation and learning. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1989.

Tony Peter Polichroniadis. *High Level Control of Virtual Actors*. PhD thesis, University of Cambridge, 2001.

Joachim Pouderoux, Jean-Christophe Gonzato, Ireneusz Tobor, and Pascal Guitton. Adaptive hierarchical rbf interpolation for creating smooth digital elevation models. In *GIS '04: Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pages 232–240, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-979-9.

Arcot J. Preetham, Peter Shirley, and Brian E. Smits. A practical analytic model for daylight. In *SIGGRAPH*, pages 91–100, 1999.

Przemyslaw Prusinkiewicz. Simulation modeling of plants and plant ecosystems. *Commun. ACM*, 43(7):84–93, 2000. ISSN 0001-0782.

Przemyslaw Prusinkiewicz. Modeling and Vizualisation of Biological Structures. In *Proceeding of Graphics Interface '93*, pages 128–137, May 1993.

Przemyslaw Prusinkiewicz. Score generation with l-systems. In *Proceedings of the 1986 International Computer Music Conference*, pages 455–457, 1986.

Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc., New York, NY, USA, 1990. ISBN 0-387-97297-8.

Przemyslaw Prusinkiewicz, Mark James, and Radomír Měch. Synthetic topiary. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 351–358, New York, NY, USA, 1994. ACM Press. ISBN 0-89791-667-0.

Przemyslaw Prusinkiewicz, Mark Hammel, Jim Hanan, and Radomr Měch. Visual models of plant development, 1996.

Long Quan, Ping Tan, Gang Zeng, Lu Yuan, Jingdong Wang, and Sing Bing Kang. Image-based plant modeling. *ACM Trans. Graph.*, 25(3):599–604, 2006. ISSN 0730-0301.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. ISBN 3-900051-07-0.

Joseph Raphson. Analysis aequationum universalis, 1690.

Robert J. Renka. Multivariate interpolation of large sets of scattered data. *ACM Trans. Math. Softw.*, 14(2):139–148, 1988. ISSN 0098-3500.

Patrick Reuter, Pierre Joyot, Jean Trunzler, Tamy Boubekeur, and Christophe Schlick. Reconstructing implicit surfaces with sharp edges via enriched reproducing kernel particle approximation. Technical report, LaBRI, 2004.

Craig W. Reynolds. Computer animation with scripts and actors. In *SIGGRAPH '82: Proceedings of the 9th annual conference on Computer graphics and interactive techniques*, pages 289–296, New York, NY, USA, 1982. ACM Press. ISBN 0-89791-076-1.

Charles Rose, Michael F. Cohen, and Bobby Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18(5):32–41, 1998.

Duncan Rowland and Frank Biocca. Evolutionary co-operative design between human and computer: implementation of "the genetic sculpture park". In *VRML '00: Proceedings of the fifth symposium on Virtual reality modeling language (Web3D-VRML)*, pages 75–79, New York, NY, USA, 2000. ACM Press. ISBN 1-58113-211-5.

Marie Samozino, Marc Alexa, Pierre Alliez, and Mariette Yvinec. Reconstruction with voronoi centered radial basis functions. In *ACM/EG Symposium on Geometry Processing*, 2006.

Jens Schneider, Tobias Boldte, and Ruediger Westermann. Real-time editing, synthesis, and rendering of infinite landscapes on GPUs. In *Vision, Modeling and Visualization 2006*, 2006.

Joshua Schpok, Joseph Simons, David S. Ebert, and Charles Hansen. A real-time cloud modeling, rendering, and animation system. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 160–166, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association. ISBN 1-58113-659-5.

Nicol N. Schraudolph and Richard K. Belew. Dynamic parameter encoding for genetic algorithms. *Mach. Learn.*, 9(1):9–21, 1992. ISSN 0885-6125.

Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, mar 1978. ISSN 0090-5364.

Cignoni Montani Scopigno. Dewall: A fast divide conquer delaunay triangulation algorithm in $\mathbb{E}^d$. *Computer-Aided Design*, 30(5):333–341, April 1998.

David W. Scott. *Multivariate Density Estimation*. Wiley, 1992.

Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524, New York, NY, USA, 1968. ACM Press.

Ilya Shlyakhter, Max Rozenoer, Julie Dorsey, and Seth Teller. Reconstructing 3d tree models from instrumented photographs. *IEEE Computer Graphics and Applications*, 21(3):53–61, May/June 2001. ISSN 0272-1716.

Robin Sibson. A brief description of natural neighbour interpolation. In V. Barnet, editor, *Interpreting Multivariate Data*, pages 21–36. Wiley, 1981.

Side Effects Software Inc. Beowulf. `http://www.sidefx.com/index.php?option=com_content&task=view&id=1041&Itemid=68`, 2008a.

Side Effects Software Inc. Houdini. `http://www.sidefx.com`, 2008b.

Side Effects Software Inc. Spider-man 3. `http://www.sidefx.com/index.php?option=com_content&task=view&id=733&Itemid=68`, 2008c.

Karl Sims. Artificial evolution for computer graphics. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 319–328, New York, NY, USA, 1991. ACM Press. ISBN 0-89791-436-8.

Alvy Ray Smith. Plants, fractals, and formal languages. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 1–10, New York, NY, USA, 1984. ACM Press. ISBN 0-89791-138-5.

Lindsay I. Smith. A tutorial on principal components analysis. `http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf`, 2002.

Elias M. Stein and Rami Shakarchi. *Real Analysis: Measure Theory, Integration, and Hilbert Spaces*. Princeton University Press, March 2005. ISBN 0691113866.

Phil Thompson. Organised chaos. `http://fractalmusician.com/`, 1999.

Ireneusz Tobor, Patrick Reuter, and Christophe Schlick. Efficient reconstruction of large scattered geometric datasets using the partition of unity and radial basis functions. In *WSCG (Winter School of Computer Graphics)*, feb 2004.

Munetoshi Unuma, Ken Anjyo, and Ryozo Takeuchi. Fourier principles for emotion-based human figure animation. *Computer Graphics*, 29(Annual Conference Series):91–96, 1995.

Sethu Vijayakumar, Aaron D'souza, and Stefan Schaal. Incremental online learning in high dimensions. *Neural Comput.*, 17(12):2602–2634, 2005. ISSN 0899-7667.

Niels Fabian Helge von Koch. Une méthode géométrique élémentaire pour l'étude de certaines questions de la théorie des courbes planes. *Acta Math*, 30:145–174, 1906.

Georgy Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *Journal für die Reine und Angewandte Mathematik*, 133:97–178, 1907.

Niniane Wang. Realistic and fast cloud rendering. *journal of graphics tools*, 9(3):21–40, 2004.

Jason Weber and Joseph Penn. Creation and rendering of realistic trees. In *SIGGRAPH 95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 119–128, New York, NY, USA, 1995. ACM Press. ISBN 0-89791-701-4.

Edward J. Wegman. Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association*, 85(411):664–675, September 1990.

Joseph Weizenbaum. Eliza — a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, 1966. ISSN 0001-0782.

Holger Wendland. *Scattered data approximation*. Cambridge University Press, 2005. ISBN 0-521-84335-9.

Paul John Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Department of Applied Mathematics, Harvard University, Cambridge, MA, 1974.

David H. Wolpert and William G. Macready. No free lunch theorems for search. Working Papers 95-02-010, Santa Fe Institute, February 1995.

David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.

Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. Instant architecture. *ACM Trans. Graph.*, 22(3):669–677, 2003. ISSN 0730-0301.

Steven Worley. A cellular texture basis function. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 291–294, New York, NY, USA, 1996. ACM. ISBN 0-89791-746-4.

Yasunori Yanai and Minoru Okada. A study on a geometry transformation method for a geometry and scene modeler by a verbal interface. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 221–230, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-564-9.

Xiao-Hu Yu and Guo-An Chen. Efficient backpropagation learning using optimal learning rate and momentum. *Neural Netw.*, 10(3):517–527, 1997. ISSN 0893-6080.

Hans-Peter Seidel Yutaka Ohtake, Alexander Belyaev. Multi-scale and adaptive cs-rbfs for shape reconstruction from cloud of points. In *Advances in Multiresolution for Geometric Modelling*, pages 143 – 154. Springer, Cambridge, UK, 2005.

Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. Sketch: an interface for sketching 3d scenes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 163–170, New York, NY, USA, 1996. ACM Press. ISBN 0-89791-746-4.

Howard Zhou, Jie Sun, Greg Turk, and James M. Rehg. Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):834–848, July/August 2007.